AFRL-RI-RS-TR-2016-013

A COORDINATED CONTROL ARCHITECTURE FOR DISASTER RESPONSE ROBOTS

TRACLABS, INC

MARCH 2016

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2016-013 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
GREGORY P. HORVATH
Work Unit Manager

/ S /
MICHAEL J. WESSING
Deputy Chief, Information Intelligence
Systems and Analysis Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JAN 2016	FINAL TECHNICAL REPORT	SEP 2012 – AUG 2015

4. TITLE AND SUBTITLE

A COORDINATED CONTROL ARCHITECTURE FOR DISASTER RESPONSE ROBOTS

5a. CONTRACT NUMBER
FA8750-12-C-0288

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702E

6. AUTHOR(S)

Robert R. Burridge, Karan Khokar, Patrick Beeson, James Kramer, Mars Chu, Frank Weng, Seth Gee, Joshua James, Robert Platt, Stephen Hart, Mordechai Rynderman, Eric Huber

5d. PROJECT NUMBER
ROBO

5e. TASK NUMBER
PR

5f. WORK UNIT NUMBER
03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TRACLabs, Inc.
8610 N. New Braunfels Street Suite 110
San Antonio, TX 78217

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIED
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2016-013

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Team TRACLabs participated in the DARPA Robotics Challenge as a Track B team (developing software only). Keeping the human operator in the loop as much as possible, and using off-the-shelf components as a foundation, TRACLabs wrote software for robot control, network management, and the operator control station. In the Virtual Robotics Challenge, Team TRACLabs placed fourth of 22 teams, earning the use of an Atlas hydraulic humanoid for the rest of the competition. In the DRC Trials, Team TRACLabs placed sixth of 16 teams, earning additional funding through the Finals. In the DRC Finals, Team TRACLabs placed ninth of 23 teams. In this report, the competitions are described, along with details on Team TRACLabs' methods and results.

15. SUBJECT TERMS
DARPA Robotics Challenge; Humanoid Robotics; Disaster Relief

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**GREGORY HORVATH**
U	U	U	UU	94	**19b. TELEPHONE NUMBER (*Include area code*)** **(315) 330-7969**

TABLE OF CONTENTS

LIST OF FIGURES

1 SUMMARY

From September, 2012, through August, 2015, Team TRACLabs participated in the DARPA Robotics Challenge (DRC). DARPA created the DRC in order to spur development of robotic technologies that will be useful in disaster relief and recovery efforts. In order to maximize participation from teams with various capabilities, four possible tracks for involvement were offered: two funded (A&B), two not (C&D); two involving Atlas robots (B&C), two using their own robot designs (A&D). Of these, TRACLabs entered "Track B", which comprised software teams that received government funding to develop user interfaces and robot control methods for controlling an Atlas humanoid robot.

In June, 2013, Tracks B and C competed to control a virtual Atlas in a cloud-based simulated environment. This Virtual Robotics Challenge (VRC) tested the teams with five runs each of three different tasks: hose (manipulation), terrain (locomotion), and vehicle (both). Network communications were degraded so that teams had to rely on a combination of autonomy and operator control. The TRACLabs entry treated the VRC as primarily a human-robot interaction (HRI) problem, embedded in a systems integration (SI) problem. By enabling the human operator to make all high-level decisions, and using off-the-shelf software where possible, we were able to focus resources on critical capabilities, such as network management. Team TRACLabs scored 30 out of 60 points, placing 4th of the 22 teams that qualified to participate in the VRC. The top six teams got further DARPA funding and an Atlas robot for the next round of the competition.

In December, 2013, nine Track A and seven Atlas teams competed in Miami, FL, using "live" robots in the DRC Trials. Each team attempted eight challenge tasks, each of which had three subtasks. Accomplishing all three subtasks without a robot fall earned a bonus point on that task. Team TRACLabs spent much of the time between the VRC and the DRC Trials learning how to house, use, and control a 350-pound hydraulic humanoid. We continued our reliance on HRI and SI, using walking and balancing behaviors created by Boston Dynamics (BDI), developing good algorithms for manipulator control, and developing graphical user interfaces that provided good situational awareness to the human operator in order to make informed strategic decisions. Once again, strong network management was a critical part of our success, as we placed 6th of the 16 teams that qualified – good enough to earn another 18 months of funding and compete in the DRC Finals.

In June, 2015, 23 teams met in Pomona, CA, to compete in the DRC Finals. These included many new international entries from Korea, Japan, and Europe. The Finals required teams to attempt all eight challenge tasks in a row in one hour, which was much more stringent than the Trials. The robots had to be tetherless and without a safety rope, which was particularly troubling for the Atlas teams. During the time between the Trials and the Finals, the seven Atlas robots were redesigned to remove the power and data tether, add a degree of freedom in each arm, and strengthen the back and legs. Team TRACLabs completely redesigned the operator interface, developed much better control of the arms, collaborated with both IHMC and MIT, and designed working solutions to all eight challenge tasks. Due to the nature of the new network shaping, we developed stronger autonomy for the robot, although maintaining the ability for the operator to interrupt execution and adjust parameters when desired. TRACLabs placed 9th out of the

23 teams at the Finals, and 4th of seven Atlas teams. We were the highest-ranking Atlas team to use BDI's controllers for walking and balancing.

We learned a lot about humanoid robotics during this three-year project, and produced some useful techniques that should push forward the boundary of the discipline. We developed relationships with scientists from other teams that we hope will grow stronger in the future as we all look for ways to collaborate and nurture this new field of robotics toward maturity. This report chronicles our process of development through the VRC, DRC Trials, and DRC Finals, discussing the technologies we used and the strategies we employed to score so highly in each competition.

2 INTRODUCTION

In 2012, the Defense Advanced Research Projects Agency (DARPA) introduced the DARPA Robotics Challenge (DRC). This program encouraged the development of technologies for semi-autonomous robots capable of service in disaster scenarios. Specifically, "to advance the current state of the art in the enabling technologies of supervised autonomy in perception and decision-making, mounted and dismounted mobility, dexterity, strength, and platform endurance" in "complex tasks in dangerous, degraded, human-engineered environments" [DARPA, 2012]. The DRC is the latest in a series of DARPA-sponsored robotics competitions that began with the "Grand Challenges" for autonomous driving in 2004 and 2005, and the Urban Challenge in 2007. Unlike those competitions, however, which required fully autonomous robots, the DRC encouraged teams to keep a human operator "in the loop", sharing autonomy between the robot and the operator.

The DRC involved three major rounds of competition for software teams: The Virtual Robotics Challenge (VRC), which took place online in June, 2013, the DRC Trials, which took place in Homestead, FL, in December 2013, and the DRC Finals, which took place in Pomona, CA, in June 2015. In this report, we do not further motivate the DRC, since that has been done eloquently by DARPA already. Instead, the rest of this introductory section is devoted to describing the components of the DRC that influenced design decisions by Team TRACLabs. Section 3 details those design decisions in the context of the three rounds of competition. Section 4 describes the results of Team TRACLabs' entry into the competitions, and Section 5 contains concluding remarks.

2.1 Four Tracks

In order to maximize participation in the DRC, DARPA created four separate tracks:

- Track A: Teams funded by DARPA to produce complete solutions (hardware and software).
- Track B: Teams funded by DARPA to produce software only (operator interfaces, software architecture, and robot control).
- Track C: Software-only, but without government funding
- Track D: Complete solutions, but without government funding.

Tracks A and B started receiving funding in September, 2012.

Tracks B and C competed in the VRC in June, 2013. The top six teams from that competition (regardless of Track B or C) were given DARPA funding and an Atlas robot to compete in the DRC Trials. From that point on, they were referred to as "Track B/C". For various reasons, seven Track B/C teams received partial or full DARPA funding and competed with an Atlas in the DRC Trials.

The top eight teams from the DRC Trials received continued government funding to proceed to the DRC Finals in June, 2015. Although only five Atlas teams placed in the

top eight, several of the next-ranked teams recombined, some funding was rearranged, and all seven Atlas teams proceeded to the DRC Finals.

After the DRC Trials, the most logical grouping of the teams was between those with an Atlas and those with a different robot. DARPA aggressively pursued participation from abroad, so that although some of the sixteen teams dropped out after the DRC Trials (for instance, NASA-JSC's Valkyrie), 25 teams qualified for the DRC Finals, including seven Atlas teams and 18 others. Of these, 23 competed in the DRC Finals in Pomona.

2.2 Eight Challenge Tasks

Based on analysis of the disaster environment at Fukushima, DARPA chose eight representative challenge tasks. Together, these tasks exercise many aspects of robot capability, including manipulation, locomotion (both mounted and unmounted), decision-making, strength, and perception. Loosely, these tasks are:

- Vehicle: The robot must enter a vehicle, drive it to a pre-determined destination, and egress the vehicle.
- Terrain: The robot must traverse rough terrain.
- Debris: The robot must clear debris from a doorway.
- Door: The robot must open a door and go through the doorway.
- Ladder: The robot must climb a ladder or staircase.
- Wall: The robot must cut through a wall using a tool.
- Hose: The robot must acquire a hose nozzle, transport it, and connect it to a standpipe.
- Valve: The robot must turn a valve.

These eight tasks formed the basis for all three competitions: the VRC, DRC Trials, and DRC Finals.

2.3 Atlas Overview

Atlas is a hydraulic humanoid robot developed for the DRC by Boston Dynamics, Inc. (BDI). Its design is a derivative of BDI's Petman robotic platform, and it underwent improvements and modifications for most of the duration of this competition.

2.3.1 Hardware

At the start of the competition, Atlas was nearly 2 meters tall and weighed about 150 kg. It had 28 actuated joints: two six-degree-of-freedom (DOF) arms, two 6-DOF legs (three in hip, one in knee, and two in ankle), three in the back, and one "tilt" DOF in the neck. It has two flat feet. By the end of the competition, much of the hardware had been upgraded or replaced, and the two distal hydraulic joints on each arm had been replaced by three electric joints.

The original, virtual Atlas used in the VRC is shown in Figure 1. Note the highly exposed head and lack of internal "organs".

For manipulation, 3-fingered hands from iRobot and 4-fingered hands from Sandia National Labs were offered initially. As the competition progressed, DARPA eased restrictions and many teams designed or purchased their own grippers. Based on this, DARPA provided passive hooks for the Trials and additional active grippers from Robotiq and SRI for the Finals. Simulated Sandia hands are also shown in figure 1.

The first (Trials) live version of Atlas had a large tether that provided power, data, and cooling water to the robot from a large blue box/radiator situated nearby. Most people felt that these robots could not sustain or recover from a fall, so they were kept on a safety belay rope at all times. If the robot fell and applied tension to the rope, an "intervention" was called, which cost five minutes and returned the robot to a reset area. In the event of a scoring tie, the team with fewer interventions was ranked higher.

The second (Finals) version had a battery pack, wireless communication, and on-board cooling, and thus no power&data tether. However, teams were not given batteries for development and had to use an off-board battery emulator, which supplied power to the robot via a power tether with a diameter of about an inch.

BDI distributed shoulder and chest pads to the Atlas teams because safety belays were not allowed during the Finals. Unfortunately, the shoulder pads widened the robot enough that it could not walk forward through the door, so few teams used them.

2.3.2 Sensors

Wrists and ankles are equipped with 3-axis and 6-axis force-torque sensors, respectively. An inertial measurement unit (IMU) mounted to its pelvis provides robot pose information. All joints provide position feedback – some provide it for both before and after the transmission.

The Multisense-SL head, developed by Carnegie Robotics, Inc., provided most of the non-proprioceptive sensing. The unit contains illuminators, its own IMU, a LIDAR single-scan laser scanner on a controllable continuously-rotating spindle, and a stereo pair of cameras. It also contains an FPGA for stereo calculations that produces depth maps.

A pair of side-facing fisheye cameras was added to the sides of the robot. The images from these cameras could be used by a human operator for situational awareness, but were too warped for most machine vision techniques.

2.3.3 Software

BDI provided low-level Atlas control code and exposed an Application Programming Interface (API) for teams to connect their software to the robot. Everybody had to use this, as it provided basic access to starting and stopping the system, proprioceptive sensor data, and joint commanding.

BDI also provided optional balancing and walking capabilities.. Some teams chose to use these behaviors, whereas others chose to develop their own "full body" controllers and handle balancing and walking in their own way.

BDI did not expose the source code for any of their software.

The head and the hands each had their own API that was independent from BDI's API. Most of these used the Robot Operating System (ROS) as their basic middleware.

2.4 Virtual Robotics Competition (VRC)

The VRC took place from June 18th to June 20th, 2013. In order to compete in the VRC, teams had to first qualify by demonstrating several simple capabilities in the virtual environment by May 15th.

The VRC exercised three of the eight challenge tasks: Vehicle, Terrain, and Hose. Software teams (Tracks B and C) had to develop operator control stations and robot control code that could successfully cause a simulated Atlas robot to accomplish these tasks within DARPA-provided virtual worlds. DARPA expected that the teams that performed well in the VRC should be able to transition their software to a physical Atlas robot fairly quickly.

Each team performed five 30-minute runs of each of the three tasks for a total of 15 independent runs. The run configurations differed by small changes to the simulated world and significant changes to the communications restrictions.

In every run, the robot started in an identical "starting pen". For two minutes, there was no network shaping and teams could perform initialization routines. They could not sense the configuration of the world until the robot had walked out of the starting pen.

A run would be terminated due to "robot damage" if the center-of-gravity of the robot sustained three or more acceleration events characteristic of a fall.

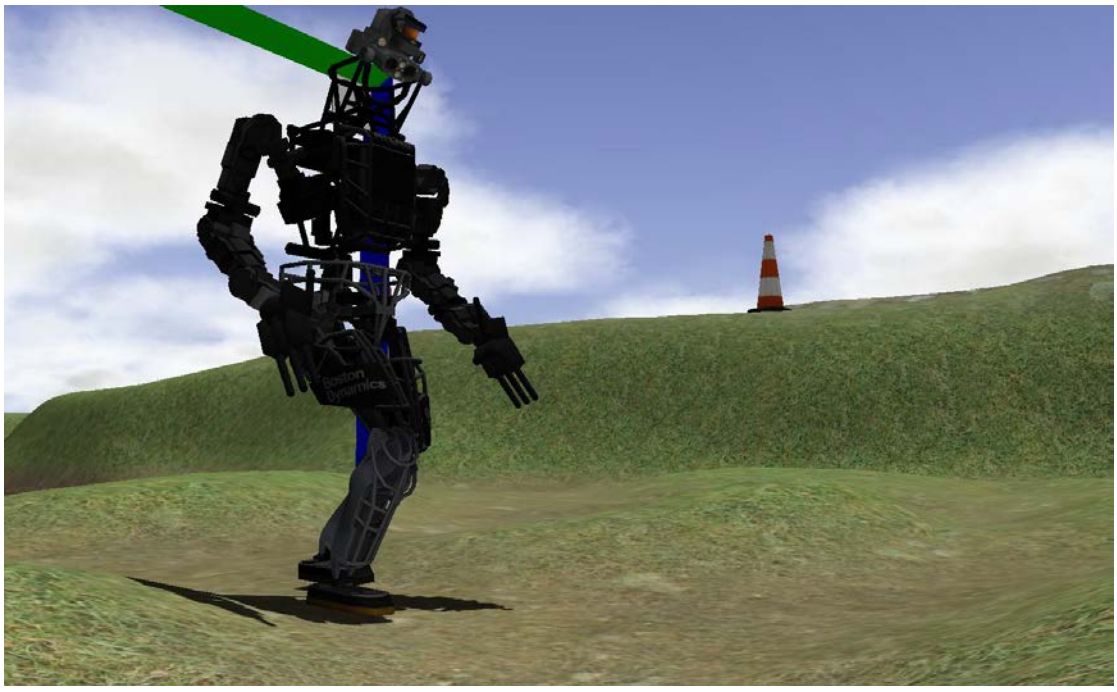

Figure 1: Virtual Atlas in the "hills" segment of the Terrain task of the VRC.

2.4.1 Virtual Atlas overview

The virtual Atlas is shown in Figure 1. This simulated robot had all expected degrees of freedom and exposed BDI's full API, including BDI's balancing and walking behaviors.

2.4.2 Communications and computer setup

For each run of the VRC, four computers were allocated in Amazon's cloud. The "Sim" computer ran the simulated world (and robot), and was not accessible to the teams. Two "Field Computers" were available for teams to upload whatever software they wished – control or perception processing or strategic. The Field computers could interact with the robot on the Sim computer via the established robot API. The fourth cloud computer was a network shaper that introduced 500ms of latency into all communication between the team's Operator Control Station (OCS) and the Field computers.

The OCS could be any computer or collection of computers used by the team to interact with the Field computers, and thereby the robot.

The Network Shaper did not limit bandwidth between the OCS and Field computers, but did impose a limit to the total number of bits transmitted. Both up and down links had independent limits, and when a team reached the limit, that link was blocked. The five uplink limits ranged from 115,200 bits to 29,491,200 bits – a factor of 256. The downlink limits ranged from approximately 59Mbits to 944Mbits.

2.4.3 Gazebo simulation

DARPA hired the Open Source Robotics Foundation (OSRF) to produce the simulation environment for the VRC. OSRF started with the Gazebo 3D simulation environment being used by Willow Garage and the PR2 research community and dramatically improved it for the VRC. Gazebo is open source, includes a robust physics engine and high quality graphics, and provides a variety of APIs for developers. Figure 1 is a representative image from one of the simulated worlds of the VRC.

Despite intense efforts by OSRF, the simulator was plagued with problems and occasional unrealistic behaviors by robot and environment throughout the nine months of development leading to the VRC. Although it reached a level of maturity by the date of the competition that was adequate, the teams sometimes had to tailor their robot behaviors to the quirks of the simulation, leading to activity that would not work at all with a real robot.

Figure 2: Walking task. From left: Starting pen and flat ground, mud pit, hills, and rubble. (From VRC Tech Guide Release 1.)

2.4.4 Walking task

The Walking Task consisted of four different terrains: (1) flat ground; (2) mud pit; (3) hills; and (4) rubble. See Figure 2 for a representative aerial view of a Walking Task world. The team scores a point for each gate traversed after leaving the starting pen platform.

Although BDI's walking controller was very good at flat terrain, it was not capable of traversing these terrains without falling.

Figure 3: Aerial view of Driving Task. (From VRC Tech Guide Release 1.)

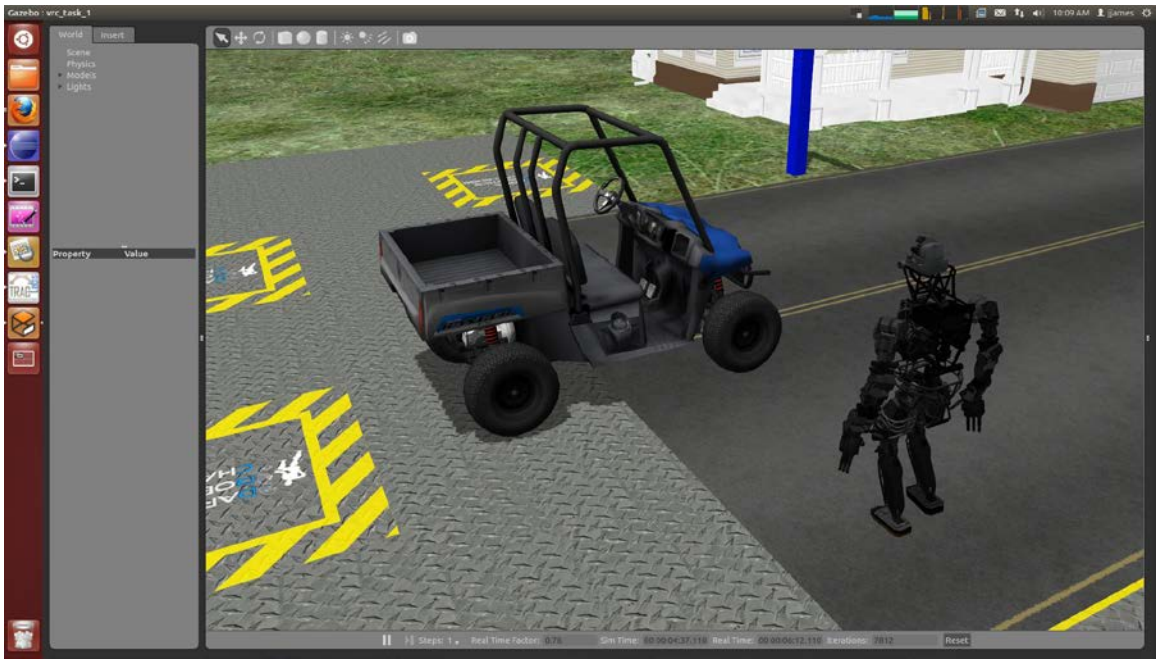

Figure 4: View of starting location of vehicle for driving task. The robot must get into the vehicle and use the steering wheel and pedals to drive it to the finish line while avoiding obstacles. (From VRC Tech Guide Release 1.)

2.4.5 Driving task

The Driving Task consisted of four subtasks: (1) to enter the vehicle; (2) to drive the vehicle along a roadway avoiding obstacles; (3) to get out of the vehicle; and (4) to walk across a finish line (See Figure 3). Each subtask earned one point.

The vehicle was modeled on the Polaris Ranger EV (See Figure 4). If the vehicle left the road or hit an obstacle, or the robot sustained "damage" (see Section 2.4), the run terminated.

Although some teams managed to create a script that got the robot into the vehicle in a manner that allowed them to drive (generally on the passenger side), the Atlas design and size relative to the Polaris made this very challenging. This concern was well-founded, as the physical Atlas hardware and Polaris were also nearly incompatible.

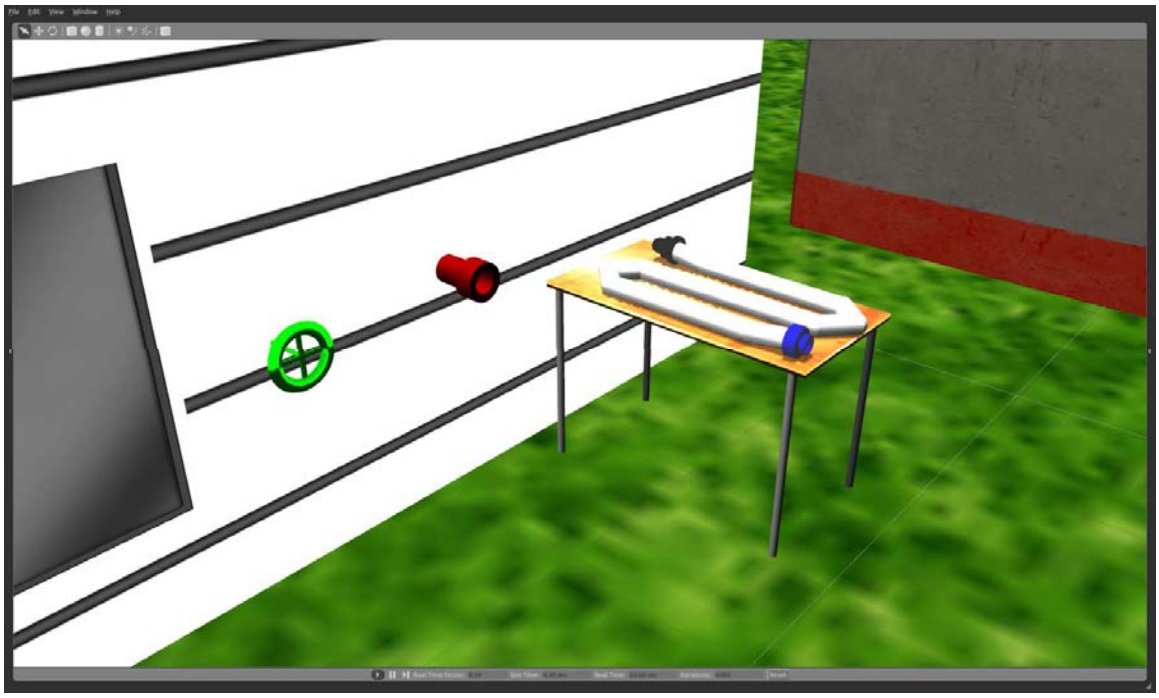

Figure 5: Closeup of Hose task, with hose, standpipe, and valve. (From VRC Tech Guide Release 1.)

2.4.6 Hose task

The Hose task consisted of four subtasks: (1) to lift the hose nozzle; (2) to mate the nozzle with a standpipe; (3) to connect the nozzle to the standpipe by threading it on; and (4) to turn a valve on the wall. (See Figure 5.) As with the other tasks, each subtask was worth one point.

This task probably strained the simulation more than the others, as Gazebo struggles with multiple complex dynamic object-to-object contact interactions.

2.4.7 Scoring

Five instances of each virtual world were created with slightly varying parameters that discouraged hard automation. Within each, four subtasks were each worth a point for a total possible score of 5*4*3 = 60 points.

2.5 DRC Trials

The DRC Trials took place in Homestead, FL, from December 17th to 21st, 2013. Teams arrived on Tuesday, Dec. 17th. Robot checkout, arena walkthroughs, comms checkout, and dress rehearsals took place Tuesday through Thursday. Each team competed once per event – eight events spread out over Friday and Saturday. Teams packed up and departed Saturday night, Dec. 21st.

Seven Track B/C teams were given "live" Atlas robots and proceeded to the DRC Trials, where they faced nine Track A teams. The Atlas teams received their robots in August of 2013, giving them approximately four months to transition from virtual to live robot and tackle the new tasks.

The Trials used all eight of the challenge tasks. Each team got one 30-minute chance to score up to three points per task, with a bonus point awarded for completing all three subtasks without falling or assistance from the field team.

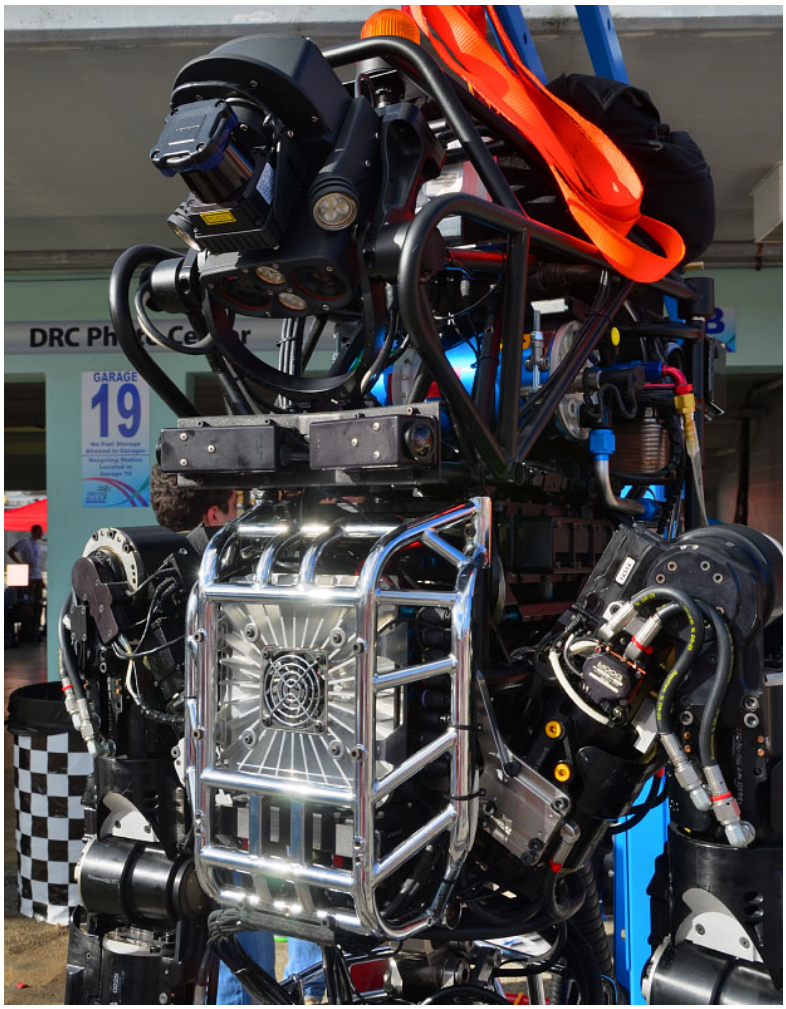

Figure 6: Closeup of Atlas torso and head. Note the upward tilted shoulder roll joints and side-facing SA cameras at the neckline.

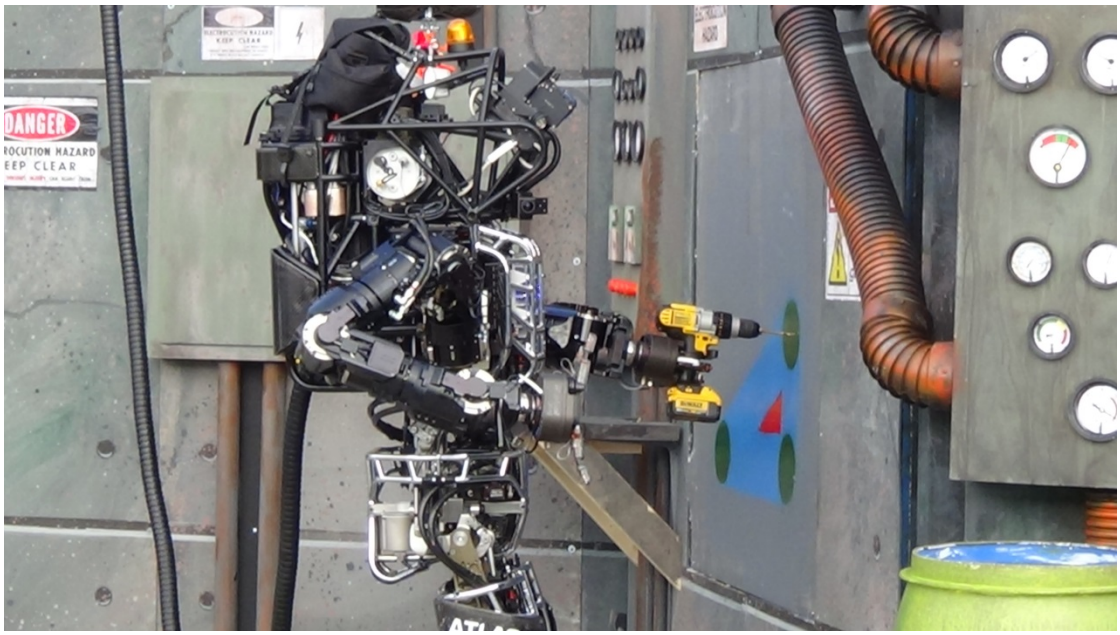

Figure 7: Side view of the Trials version of Atlas. Note the large black tether for power, data, and cooling water.

2.5.1 Atlas

Teams that earned Atlas robots were invited to Boston for Atlas training and to see the new robot in mid-July, 2013. Once the teams had demonstrated that they had the required support equipment and a lab facility that could house the robot, BDI sent them a robot and BDI techs installed it and checked it out. The Trials version of the robot is shown in Figures 6 and 7. No changes were made to the robot during the four months between delivery and DRC Trials except repairs.

Pairs of both iRobot and Sandia hands were delivered with the robot. Both are cable-driven, with the iRobot hands using strong fishing line and the Sandia hands using stranded metal cabling. It turned out that the tendons in the Sandia hands were not user-serviceable, whereas the iRobot tendons were. Furthermore, Sandia techs themselves were not able to repair a finger with a broken tendon – all they could do was replace it with a spare. For this reason, very few teams used the Sandia hands in the competition.

As the Trials approached, DARPA relaxed the restriction on what hands were permitted during the Trials, allowing teams to purchase their own, or use other end effectors of their own design. Furthermore, BDI fabricated passive hooks for all Atlas teams. Teams were allowed to change end effectors between the various tasks – provided that they carry all the end effectors with them for all the tasks. In Figure 7, the black bag on top of Atlas's head contains a collection of end effectors.

2.5.2 Communications and computers

The computer architecture for the Trials was similar to that of the VRC, except that the Field Computers were physically present in each team's "garage". A network shaper called the "Mini-Maxwell", made by InterWorking Labs, was placed between the Field computers and the OCS. A fast network connected the Field computers to the robot.

The Mini-Maxwell imposed two different communications regimens, switching every minute. In the "good comms" phase, the bandwidth was 1Mbit/sec and the latency was 100ms round trip. Under "bad comms", the bandwidth was 100kbit/sec and latency was 1000ms round trip.

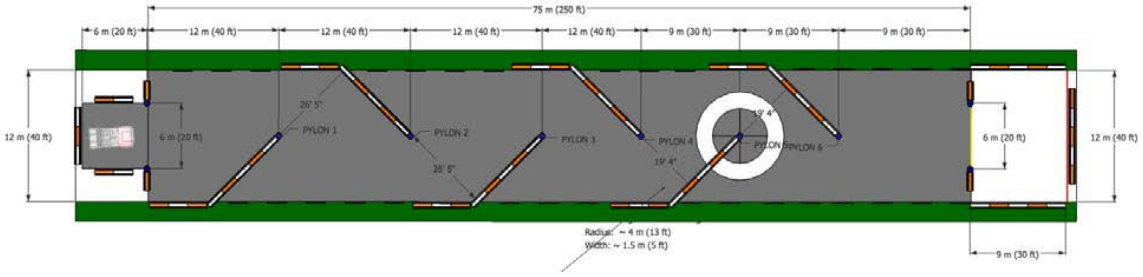

Figure 8: Trials driving course. The vehicle must slalom through the 250-foot course from the start area on the left to the end zone on the right. (From DRC Trials Task Description Release 11.)

2.5.3 Vehicle task

In the Vehicle task, robots were allowed to start in the vehicle. One point was awarded for driving the complete length of the course (See Figure 8). Two more points were given for egress and walking out of the end zone. Most teams that attempted this task decided to forego the possibility of the egress points and strap their robot into the vehicle.

This was especially true for Atlas teams, as none had found a reliable method of egressing the vehicle without a fall, and the risk of damage from a fall was too great. This is the only one of the eight tasks where Atlas robots were required to act without a safety belay rope.

Figure 9: Terrain task. The robot starts at the right and proceeds over the ramps, then the herringbone cinderblock pattern, then the flat stairs, then the slanted stairs. (From DRC Trials Task Description Release 11.)

2.5.4 Terrain task

The Terrain task consisted of progressively harder terrain for the robot to walk over. (See Figure 9.) One point was awarded when a robot traversed the ramps and herringbone pattern, another after going up and down the flat stairs, and the third after going up and down the slanted stairs. Teams could choose any path through the terrain they wanted.

2.5.5 Ladder task

For the Ladder task, teams had a choice of handrail configurations and angle of incline (60 or 75 degrees). The first point was awarded for having all points of contact at or above the first step. The second point was for the fourth step, and the final point was for all points of contact on the landing. (See Figure 10.)

2.5.6 Debris task

In the Debris task, a collection of planks of wood were arranged in a pre-defined pattern in a metal truss and placed in front of a doorway (See Figure 11). Teams earned the first point for removing any five items of debris, and the second point for another five pieces of debris. Having done this, a team could earn the third point by walking through the doorway.

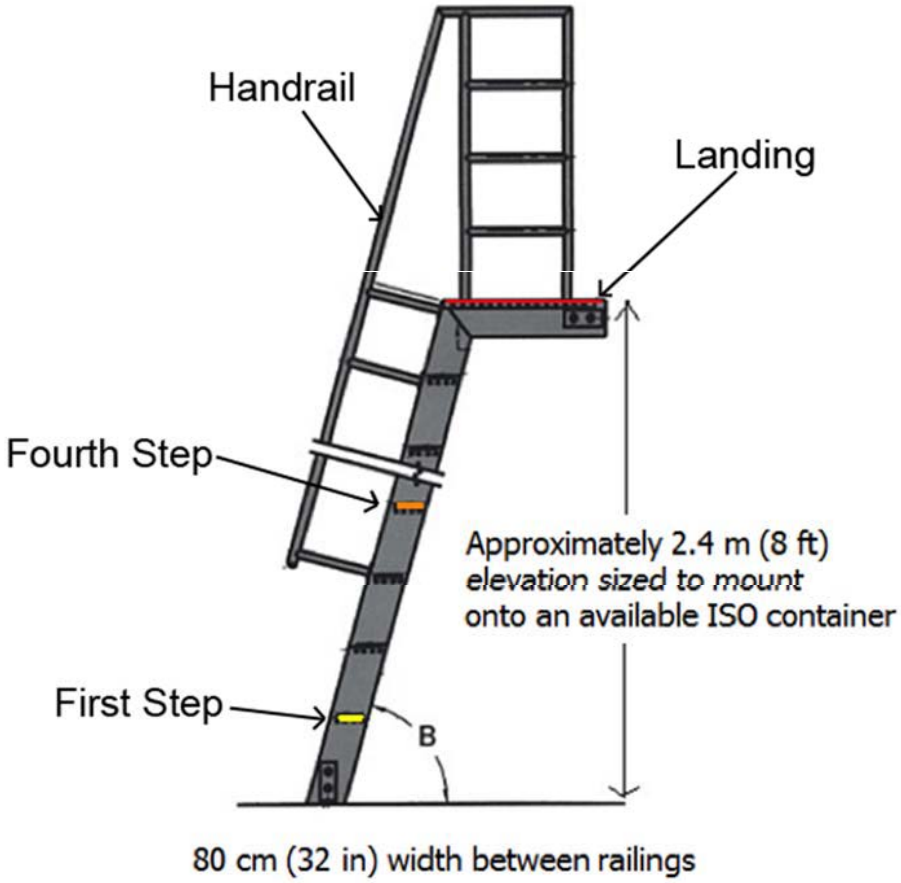

Figure 10: Ladder task. **(From DRC Trials Task Description Release 11.)**

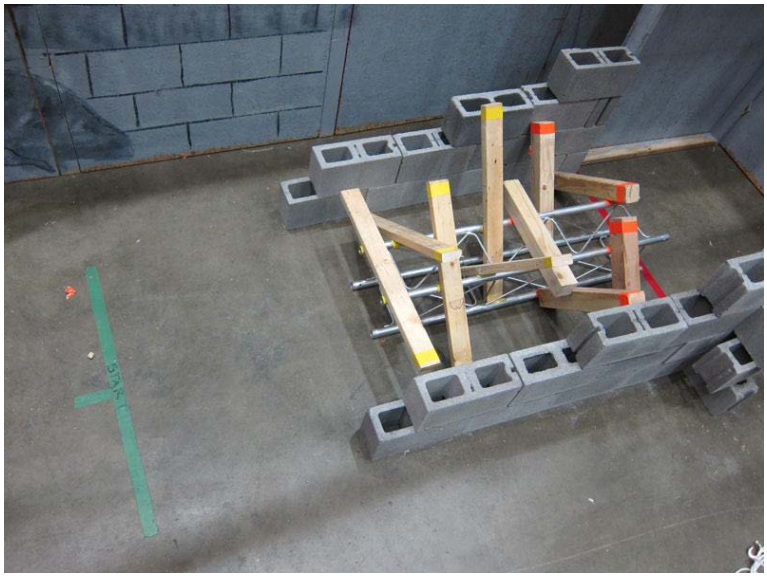

Figure 11: Debris in front of a doorway. **(From DRC Trials Task Description Release 11.)**

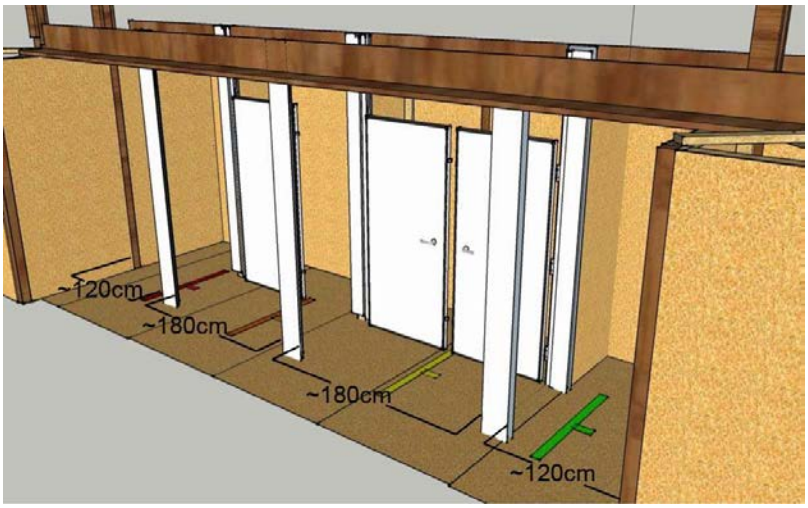

Figure 12: Door task. Robot starts at the right and must traverse three doors. **(From DRC Trials Task Description Release 11.)**

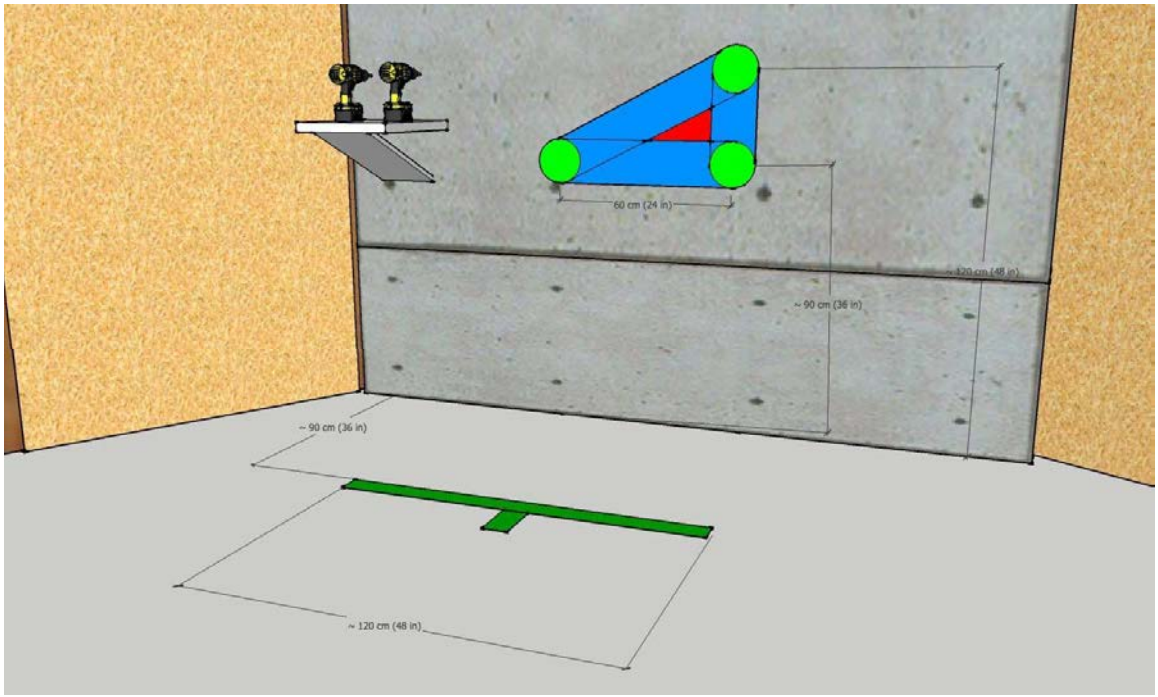

Figure 13: Wall task. The robot must pick up a drill and cut a triangle out of a (drywall) wall. **(From DRC Trials Task Description Release 11.)**

2.5.7 Door task

In the Door task, the robot must traverse three doors for one point each. The first is a push door, the second is a pull door, and the third is a pull door with a spring causing it to close. All three doors have lever handles. (See Figure 12.)

2.5.8 Wall task

In the Wall task, the robot must pick up a drill, turn it on, and use it to cut a triangular hole in a sheet of drywall. One point was awarded for each side of the triangle, unless the cut strayed beyond the boundary of the triangle. To get the third point, the robot also had to punch out the triangle. (See Figure 13.)

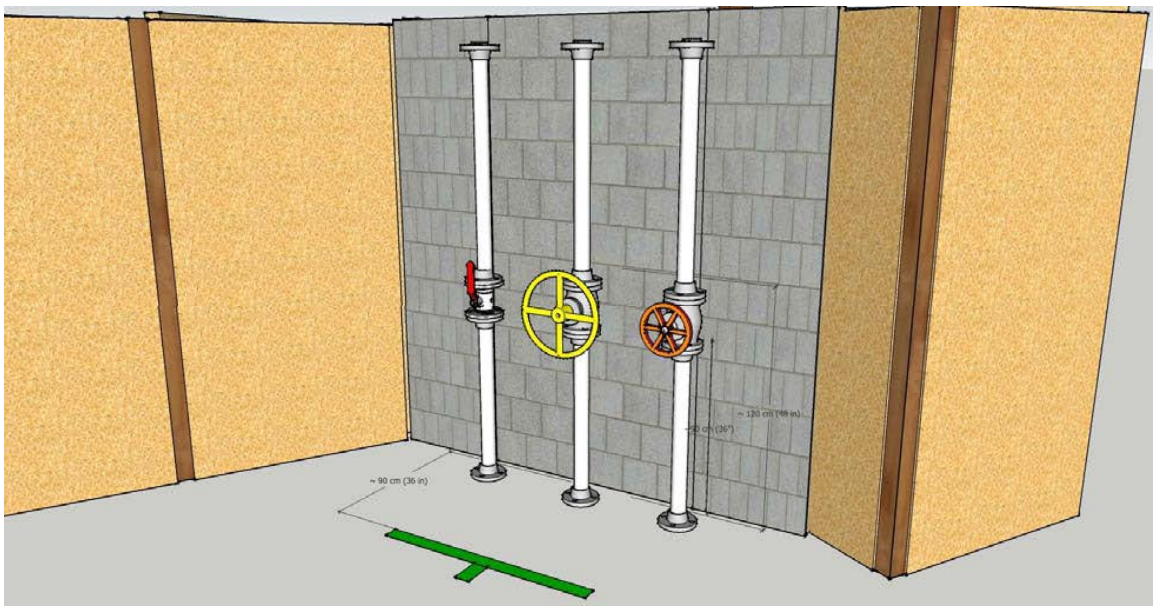

Figure 14: The Valve task consisted of three valves. One lever handle and two wheel handles. **(From DRC Trials Task Description Release 11.)**

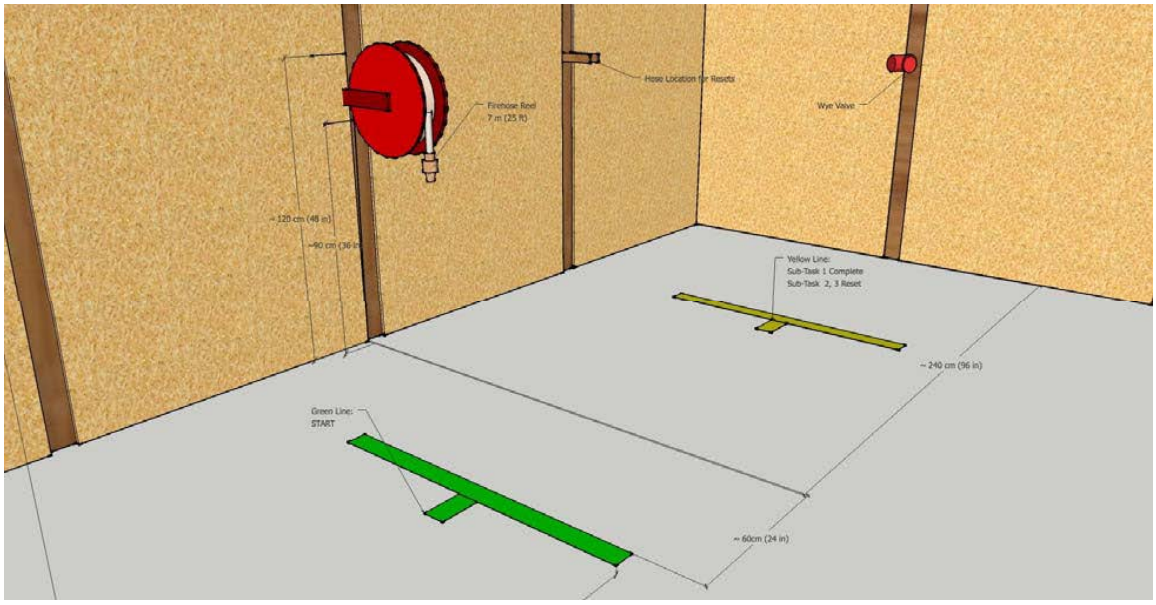

Figure 15: Hose task. The robot must acquire the nozzle on the reel on the left, walk to the standpipe on the right and connect the hose to the wye. **(From DRC Trials Task Description Release 11.)**

2.5.9 Valve task

In the Valve task, the robot must turn a lever valve 90 degrees and two circular valve handles a full revolution. Each valve handle successfully negotiated is worth one point. (See Figure 14.)

2.5.10 Hose task

The robot must acquire the nozzle of a hose on a reel, walk to a wye standpipe, and thread the hose onto the wye. The first point was awarded when the robot crossed a line on the ground while holding the nozzle. The second point was for touching the nozzle to the wye, and the third point was for threading the nozzle onto the wye. (See Figure 15.)

2.6 DRC Finals

The DRC Finals took place in Pomona, CA, during the week from May 31st to June 7th, 2015. Teams were welcomed to the event offsite on Sunday, May 31st, and got access to the garages onsite on Monday, June 1st. Vehicle shakeout, course walkthrough, comms check, and dress rehearsal took place Monday through Thursday. Competition runs were Friday and Saturday, with the better run determining the score for a team. Teams vacated the garages on Sunday, June 7th.

In the DRC Finals, the eight challenge tasks were modified: The Vehicle task was divided into Driving and Egress, the Debris and Terrain tasks were presented as a choice of one or the other, the ladder was replaced with stairs, and the Hose task was eliminated. One "surprise" task was added, which was actually one of a handful of known manipulation tasks, such as pulling a lever or pushing a button.

DARPA increased the difficulty substantially for this competition:

1. Robots were required to be tetherless, meaning wireless comm and battery only.
2. No safety belay: robots recovered from falls or ended run.
3. As many tasks as possible were performed in a single run.
4. The field team was not allowed to interact with the robot (e.g., no hand changes.)
5. Some tasks must be performed in a certain order.
6. The robot must traverse the driving course first, either by driving or walking.

The network shaping was once again different from all previous iterations. It seem much worse than the others.

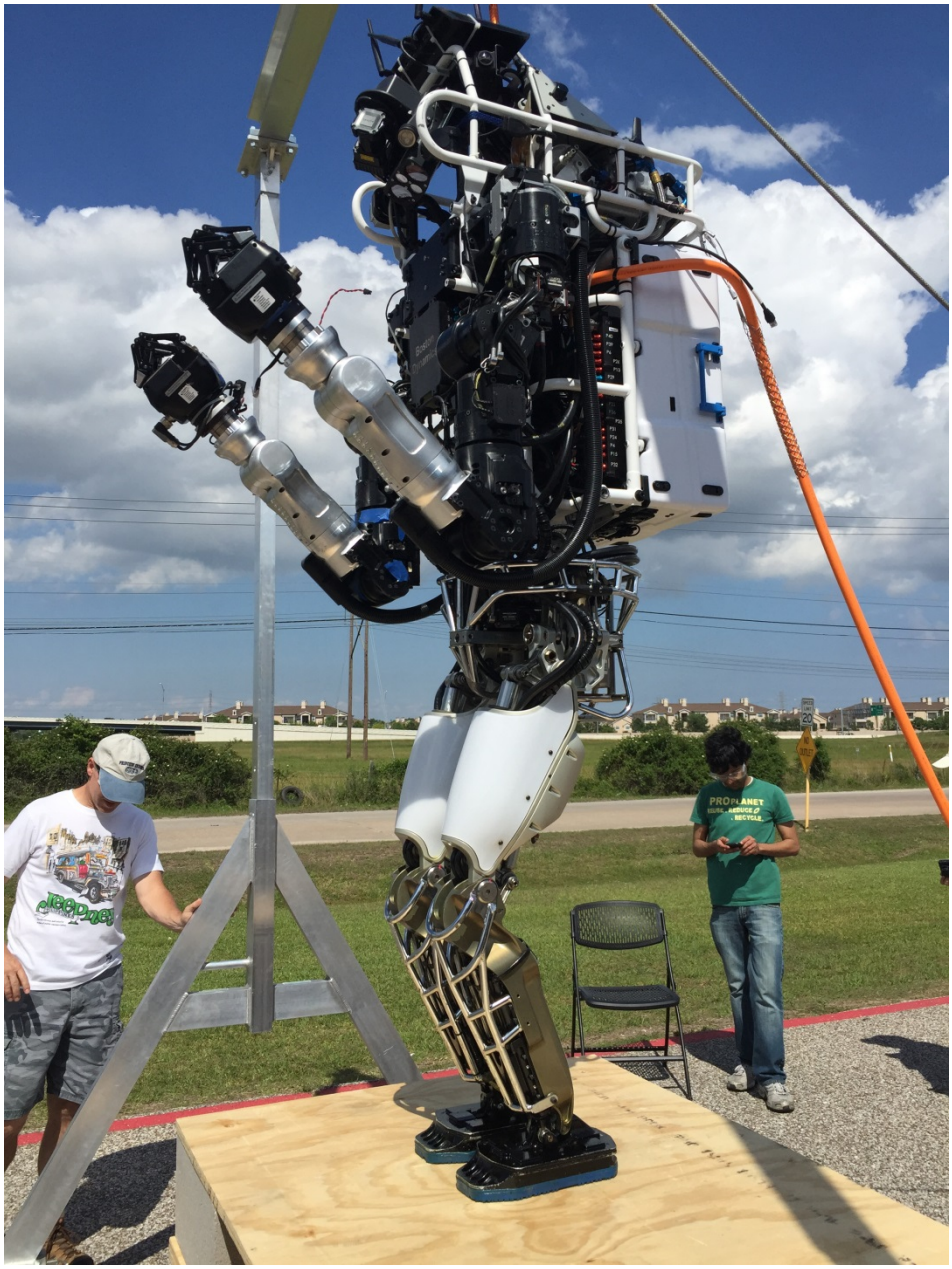

Figure 16: Final Atlas configuration for the DRC Finals. Note the battery pack and electric forearms. The orange cable provided power from an off-board battery emulator.

2.6.1 Finals Atlas overview

In the 18 months between Trials and Finals, the Atlas was significantly overhauled.

1. Many of the hydraulic actuators in the legs and back were upgraded with stronger pistons.
2. The kinematic configuration of the shoulders was modified to provide a better overlap of the dexterous workspaces of the two arms within the field of view of the Multisense head.
3. The distal two hydraulic joints in each arm were replaced with an electric forearm, which had three degrees of freedom.
4. The SA cameras were moved up to the same height as the Multisense head, and angled slightly forward.
5. Two Robotiq hands and one SRI hand were provided to all Atlas teams.
6. A wireless router was installed on-board the robot, communicating with a partner router off-board.
7. Due to potential network issues from the wireless communication, three computers were installed on the robot for high-frequency feedback control and perception signal processing.
8. The cooling system was redesigned to be entirely on-board, with a cooling fan behind the head.
9. Chest and shoulder pads were given to the teams to help protect the robot in falls. Few, if any, teams used the shoulder pads.
10. A battery pack was installed as a backpack. Due to the dangerous nature of lithium ion batteries and their schedule for design and manufacturing, teams were given an off-board battery emulator with a large orange cable to provide power to the robots. The battery pack was weighted to approximate the expected mass distribution of the true battery. Teams never operated with the actual battery except during the Dress Rehearsal and the two competition runs.
11. Teams were allowed to add their own hardware to the robot, subject to the constraint that BDI would not service it.

See Figure 16 for an image of the Finals Atlas (with orange battery emulator cable). This particular configuration has two Robotiq hands, and is lacking the chest pad that was used during the Finals.

Figure 17 shows an image of the Finals arena. The starting line is at the far end of the sand driveway on the right side. The robot may drive or walk (no drive or egress points) to the finish area, which is the paved part nearest the door. If driving, the robot may egress or be removed by a reset (no egress point). Next, the robot must open the white door and walk through it, at which point it is considered to be “indoors”.

Tasks in order from right to left are: Valve (the valve wheel is on the dark vertical pipe under the left side of the Jumbotron); Wall (two pairs of drills on left side, drywall to their right); Mystery (electric/magnetic coupler); Terrain and Rubble; and Stairs outside the exit, all the way to the right.

In Figure 18, we show the flow chart of possible task activity. In the Finals, robots had to attempt either the Terrain or Rubble tasks to get to the exit.

Figure 17: Task arena. Driving track is on the right, leading to the Door. From right to left indoors: Valve, Wall, Mystery, Terrain/Rubble, and (beyond the exit to the left) Stairs.

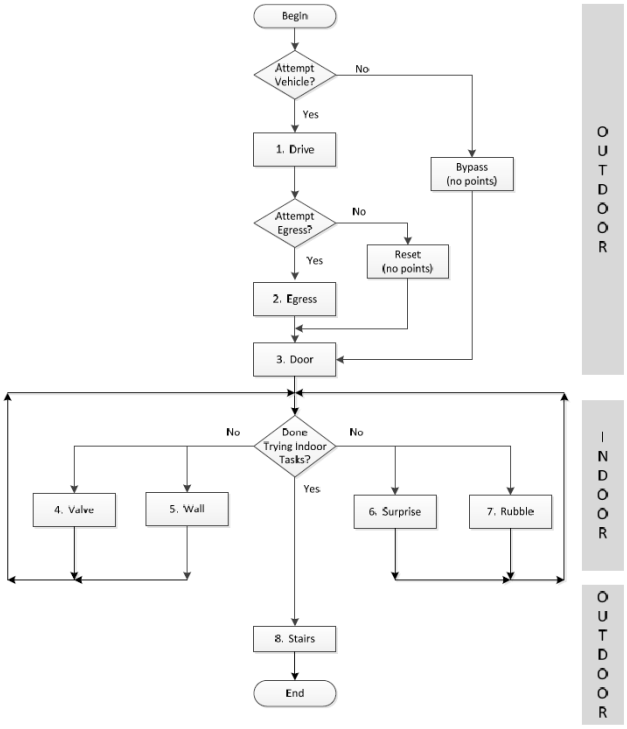

Figure 18: Robots must either drive or walk (bypass) the driving course to get to the door. Once through the door, tasks may be attempted in any order until exiting the building and attempting the Stairs task. (From DRC Finals Rule Book.)

2.6.2 Communications and computers

As noted in #6 and #7 in Section 2.6.1 above, three computers were installed on the robot. These were used for high-frequency feedback control and perception processing. Teams were also allowed one off-board Field Computer, which had unrestricted communication to the robot, albeit over a wireless link. Field Computers for all teams sat in a secure closet at the competition, with limited physical access for the teams, although they were accessible via the network.

Operator control stations were in the team garages, and could consist of one or several computers, with one or several monitors, subject to an upper bound on total power consumed.

Communication between the OCS and the Field computer/robot used two Ethernet connections: one was high bandwidth, but subject to network shaping, and the other was very low bandwidth, but was not subject to network shaping.

Communication between the OCS and the Field computer/robot was divided into two categories:

Outdoor activity consisted of the Driving, Egress, Door (up until the robot crossed the threshold), and Stairs (after the robot crossed the gate at the other end of the arena) tasks. There was no network shaping in this mode.

Indoor activity covered the Valve, Wall, Mystery, and Terrain/Rubble tasks. In this mode, the main comms link was subject to intermittent 1-second bursts of full bandwidth alternating with periods of total comms blackout that lasted up to 30 seconds. The average duration of the blackouts got shorter from 30 seconds to zero as run time progressed from Start to 45 minutes. After that, there was full communication regardless of the location of the robot.

2.6.3 Driving task

If the team was attempting the Driving task, they were allowed five minutes to modify the Polaris vehicle, provided no power tools were used. These modifications could include additions to the vehicle, such as running boards or steering mechanisms or car seats, or replacements to parts, such as the hood. Many Atlas teams had elaborate setups to help them negotiate the physical mismatch between the Atlas and the Polaris. The starting pose of the Atlas had to be such that its center-of-mass was within the body of the Polaris.

The Polaris was velocity-limited, and would not move unless the accelerator pedal was pressed, avoiding the need for braking. Reverse gear was not allowed.

The team was required to use the robot to turn the steering wheel and press the accelerator to cause the Polaris to traverse the driving track, avoiding contact with the barriers and arriving on the paved End Zone. One point was awarded for successfully crossing the line into the End Zone with the Polaris. Because the End Zone was on the edge of the Homestead track apron, there was a noticeable slant back toward the sand that the vehicle had to overcome in order to finish this task.

Once in the End Zone, the team could request that the vehicle be disabled, preventing further motion.

Teams had the option of "Bypassing" the Driving task by traversing the 200-foot sand track without the Polaris. The track consisted of compacted sand, which was watered at the start of the day, but during the day, unpredictable pockets of sand became loose, making it quite a challenge for walking or small-wheeled robots. No point was awarded for bypassing.

2.6.4 Egress task

Once the vehicle was disabled, a team could choose to try to egress the vehicle. Once all points of contact for the robot were within the marked square outside the Door, a point was awarded. The robot did not need to be standing to receive the point. Many Atlas teams had elaborate additions to the Polaris to assist in egress.

A team could choose to reset from the Vehicle, sacrificing five minutes and a chance to receive the Egress point in return for lower risk of falling. Both MIT and TRACLabs had Atlas falls on Egress in one of their runs, showing the danger of this task.

2.6.5 Door task

This task required the robot to open the door and walk through the doorway. The door had a lever handle on the left and opened away from the robot. Due to the slant of the racetrack apron, once the door was ajar, it swung completely open. A point was awarded when all points of contact were across the threshold. This was also the start of the "indoor" mode.

2.6.6 Valve task

The robot must turn the valve wheel one full revolution counterclockwise. One point was awarded upon completion.

2.6.7 Wall task

The robot must acquire one of two types of drill, activate the drill, and completely remove a circle that is marked on the drywall. A point is awarded when the segment of drywall containing the circle has been completely removed. DARPA requested that teams do "something sensible" with the drills after that, but there was no penalty for dropping the drill, nor reward for spending the effort not to.

2.6.8 Mystery task

The Mystery task changed from day to day, but was announced the previous evening. For the dress rehearsal, it was a push button. For Day One, it was a breaker box lever, and for Day Two, it was removing a magnetic connector from one receptacle and placing it in another. A point was awarded upon completion of the task.

2.6.9 Debris/Terrain task

Teams were given the choice between traversing uneven terrain consisting of a set of uneven cinderblocks, or wading through a disorganized collection of debris. Teams with walking robots tended to favor the Terrain, whereas those with wheeled robots preferred the Debris.

All robots had to accomplish one or the other before proceeding to the exit and the Stairs.

2.6.10 Stair task

After the robot crossed through the exit archway, the comms were switched back to "outdoor" mode. The robot had to climb up a set of four steps. A point was awarded when all points of contact were at or above the top platform. Contact with the railing was acceptable.

An interesting arena design choice was evident here. Whereas the End Zone for Driving, the Door, and all the indoor tasks inherited a significant slope from the racetrack apron, the Stair apparatus was levelled against gravity. This meant that there was a five to ten degree angle difference between the ground and the first step, meaning that the right end of the first step was up to about two inches higher than the left.

3 METHODS, ASSUMPTIONS, AND PROCEDURES

In this section, we detail the various techniques used by Team TRACLabs to approach the task of using an Atlas – simulated or live – to accomplish the challenge tasks. In preparing for all three competitions, our governing philosophy has incorporated two main ideas: (1) Whenever possible, rely on the human operator for high-level strategic decisions, including interpretation of complex sensor data; and (2) Use off-the-shelf solutions if they exist.

Common components of the software architecture produced by TRACLabs for each competition were:

1. User interface, including both functional and task-specific windows,
2. Network management, handling the relevant type of network shaping,
3. Robot control strategies, including inverse kinematics and trajectory planning.

3.1 VRC

We decided to rely on BDI's controllers for walking and balancing in the VRC, enabling us to focus on development of the Operator Interface, control algorithms, and manipulation activities.

3.1.1 Infrastructure

Inverse kinematics.

TRACLabs developed a novel algorithm for controlling the Atlas manipulators via Inverse Kinematics (IK). This algorithm greatly improved KDL IK solutions and speed. It used nearest neighbors from forward kinematics to speed up inverse kinematics, which makes Cartesian commanding of remote assets more tolerable.

1. Run forward kinematics for each arm offline for the entire configuration space.
   a. Arm only: fine resolution in joint space
   b. Arm + back joints: slightly coarser resolution
   c. Ignore Cartesian poses behind robot's pelvis

2. Save four hash tables (left/right arm, left/right arm + back joints) where the Cartesian pose from forward kinematics maps to all possible joint solutions.

3. Load each hash table at runtime on robot computer and populate four kd-trees with the 3D position from each Cartesian pose.

4. When IK for a left/right hand is needed, find the nearest neighbors to the desired x,y,z using the kd-tree
   a. Find union of 100 closest neighbors and all neighbors within 2cm.
   b. Always try only arms first, as back motions in Atlas are not desired while balancing during manipulation

5. Lookup all joint configurations that can reach the nearest neighbor positions

6. Run KDL

a. IK solver using the joint configurations as initial seeds KDL solver runs at least 100 times

b. KDL solver returns VERY quickly due to the small search space

c. In practice 100 solves takes ~0.1 seconds

7. Score all IK solutions

8. Any number of scoring metrics can be used

a. Penalize back motions; Penalize near limits;

i. Reward small motions

9. Use best solution

a. If no solutions exist, and the user has allowed, the robot may simply use the neighbor with the closest orientation to the desired Cartesian pose.

Communications.

We decided to use ROS for all our process communications. Because ROS has a handshaking protocol at startup for each message, there was the danger of using too many of our allotted bits during startup. To avoid this, and to keep tight control on the amount of data transferred, all messages passed through a single "gatekeeper" ROS process on each side of the shaped comm link. The Operator had the ability to set the rate at which each important message came through, with the fastest rate being 1Hz and the slowest being "pull", meaning the message was not sent unless the Operator specifically requested it.

Point Clouds.

We projected the 3D point cloud onto the two-axis planes and then compressed and sent those. On the OI side, we reconstructed the point cloud from these images. This process reduced the amount of data sent by a factor of about 100, without much loss of data. Since the human operator was interpreting the images on the OI side, this worked well for us. The compression algorithm is outlined here:

1. Full point cloud: ~11MB.

2. Local, sparsified point cloud: ~5MB.

3. Manipulation task "height" and "forward facing" 2D projections: ~300 KB together.

4. Zlib compressed 2D projections: ~52KB together. This is about 1% of the local sparsified point cloud!

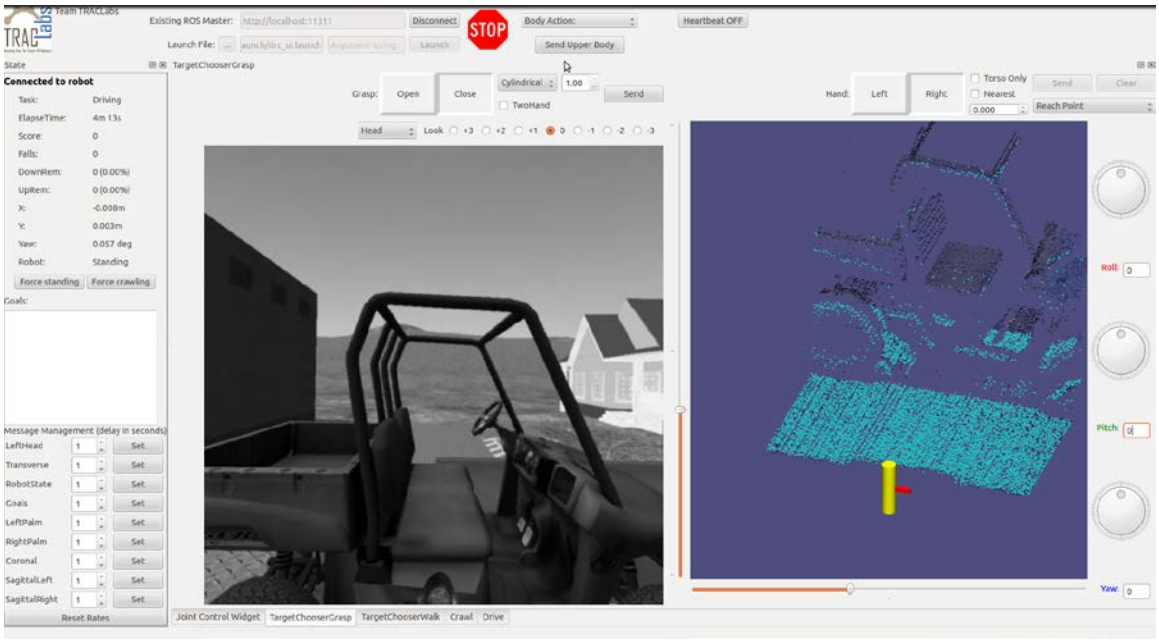

Figure 19: Operator Interface. Tab for the grasping functionality.

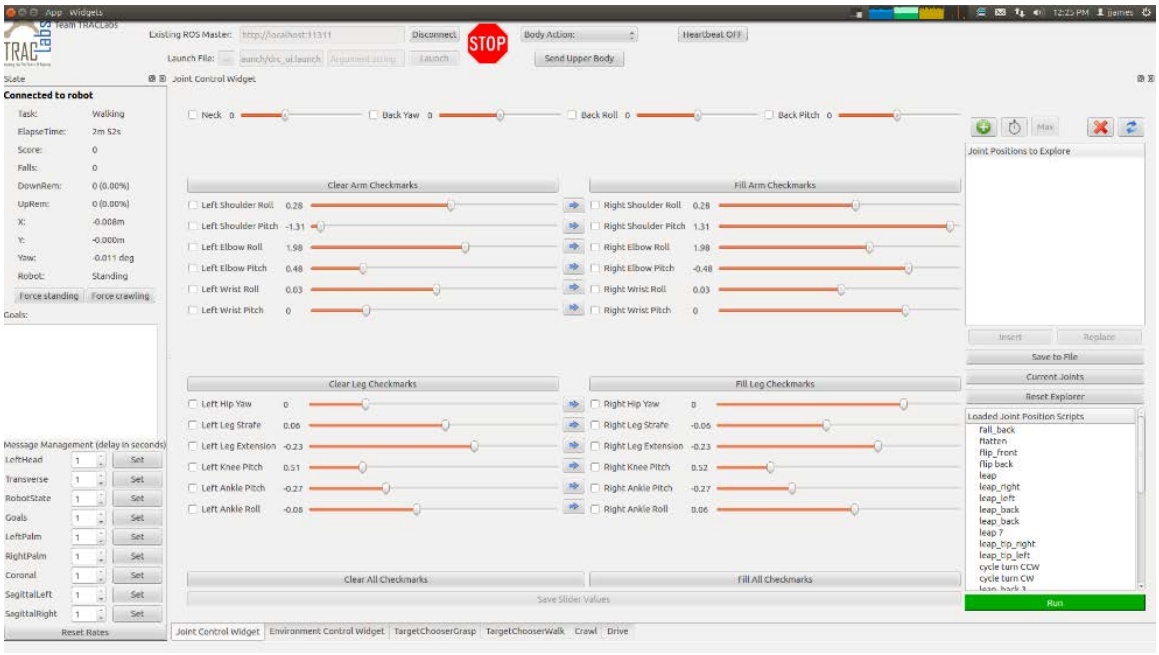

Figure 20: Joint scripting tab.

3.1.2 Operator Interface

The Operator Interface (OI) had separate tabs for the different tasks and important functionality. Figure 19 shows the tab for the grasping functionality. Figure 20 shows the joint scripting tab, where the Operator could create a precise joint-level sequence of commands and save them for later use.

The task tabs consisted of buttons that triggered appropriate joint scripts or behaviors.

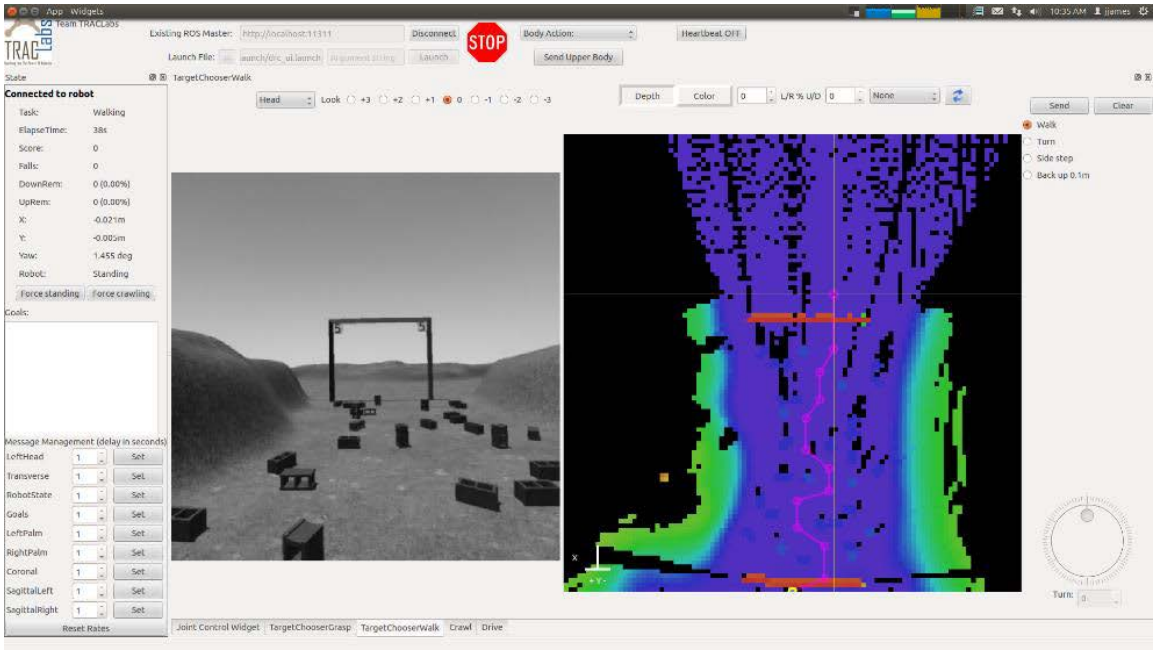

Figure 21: Operator Interface. Walking tab, showing the ability to produce a series of walking waypoints in the XY projection of the point cloud.

Always visible on the left side of the OI were some system information and widgets for setting the downlink rates for various messages. Across the top, important functionality was always available, such as halting the current action, or connect/disconnect communication. (See Figures 19 and 20.)

3.1.3 Walking task

Because BDI's walking behavior was not robust enough to traverse the different terrains in the Walking Task, we developed a set of innovative crawling algorithms that avoided falls while traversing the course, and scripts that transitioned between them smoothly.

For exiting the starting pen and walking to the first gate, BDI's walking was sufficient. We gave the Operator the ability to place waypoints in a reconstructed 3D point cloud, and the system would then produce an appropriate set of footfalls for the BDI walking controller. This capability was exposed on the "Walking tab", and was used whenever we needed to move the robot from one place to another on flat ground.

We developed a repeatable script that would cause the robot to get up from a fall. The robot moved through a sequence of poses, ultimately balancing on its feet. At that point, we could transition into BDI's STAND mode, and from there into their walking behavior.

Mud. To traverse the mud pit after the first gate, we had a behavior that caused the robot to lie down on its back without triggering a fall. From there, we had a locomotion behavior that was essentially a "dry land backstroke" that moved the robot forward, and

two slightly modified versions of it that rotated the robot clockwise and counterclockwise. Combining these, the Operator was able to drive the robot into the mud pit, through the mud, and up the slope on the other side. Although the mud was opaque from the top down, it was transparent from below, so we were able to see the first gate and use it as a reference while "swimming" through the mud.

Hills. Once we had exited the mud, we transitioned from backstroke pose to "frog" pose. This pose had the robot resting on all four limbs, supported by the fingers and toes. By executing a fast spring-like motion, the robot was able to hop forward without triggering a fall event. Again, we also created clockwise and counterclockwise versions so the Operator had full control of the trajectory.

Rubble. The frog behavior was also successful in traversing the rubble. The operator had to be careful not to leap into a standing cinderblock, as this shock to the head was sometimes enough to trigger a fall event.

The top two images in Figure 22 show the backstroke behavior entering and exiting the mud pit. The bottom two images show the frog behavior in the Hills and Rubble portions of the task.

Figure 22: Top two: Atlas entering and leaving the mud pit using the "backstroke" behavior. Bottom two: Atlas using the "frog" behavior in the hills (left) and rubble (right).

3.1.4 Driving Task

We spent much time attempting to figure out a way to get Atlas to enter the vehicle in a manner that would permit driving. However, in the end we were not able to find a script that could accomplish this task.

As a fallback, we developed a script that got the robot into the seat of the vehicle far enough that it got the first point for the task fairly consistently. To do this, the robot

walked close to the vehicle and put both arms through the roll bars – one on either side of the passenger doorway. Next, the arms moved together down and back, essentially dragging the robot off its feet and horizontally onto the seat. By curling the legs up behind the body, the pelvis moved far enough into the vehicle to trigger the point.

3.1.5 Hose task

We developed robust "operator-in-the-loop" methods for picking up the hose and for turning the valve. We developed a tab on the OI that provided the Operator with the ability to make small adjustments to the 6-DOF pose of the nozzle, but even so, with the updates coming at most once per second, found it incredibly difficult to line the nozzle up well enough to mate with the receptacle. We were not able to develop a method of threading the nozzle onto the receptacle, once it was mated.

3.2 DRC Trials

When we placed highly enough in the VRC to get an Atlas robot, we rented additional lab space to house the robot and the many challenge task apparatuses we knew we would need to develop strategies for all eight tasks. We received the robot in early August, 2013, and thus had about four months to prepare for the DRC Trials.

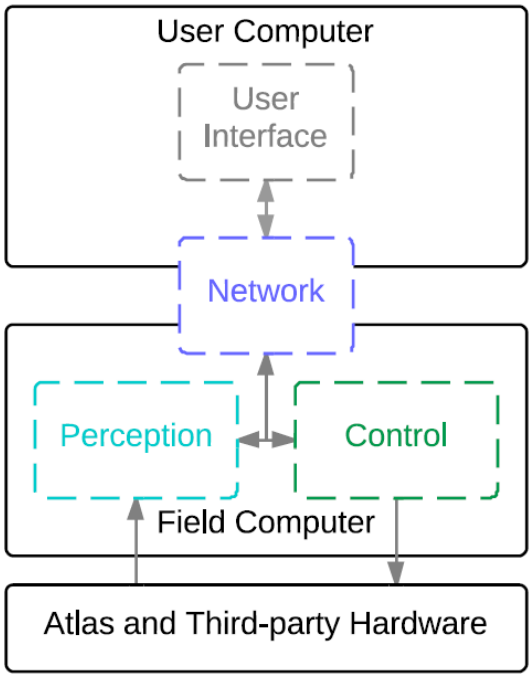

Figure 23: High-level system diagram.

3.2.1 Infrastructure

Figure 23 shows the high-level system diagram for our processes on the computers. Each of the four functional categories outlined with a dashed line has characteristic issues relevant to the DRC Trials.

3.2.1.1 Networking software

The communications throttling described in Section 2.5.2 led to some ROS handshaking problems. Communication within ROS is performed with topics, or named connections, that transmit messages of a specific data type. ROS nodes, or computational units, connect to arbitrary sets of topics that supply the requisite data for each node's operation. Nodes open connections to other nodes by querying a white-pages service called a "ROS master"; the master provides network location information and negotiates connection parameters, enabling direct connections between nodes.

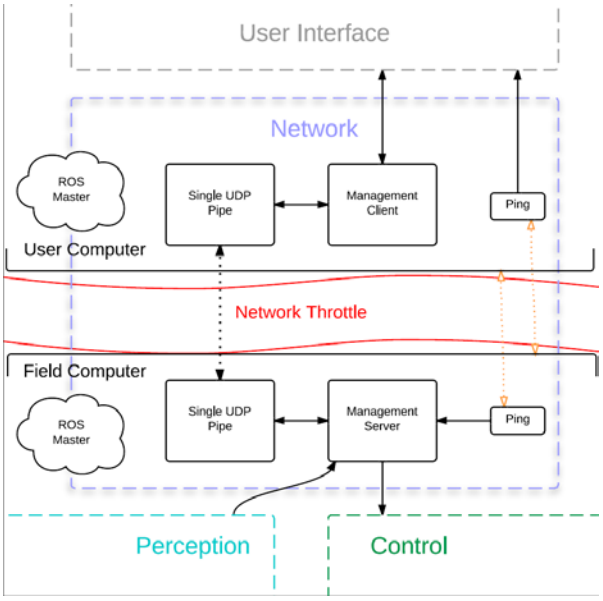

Figure 24: Network detail diagram. Each computer has its own ROS Master, which manages local connections between ROS nodes. All connections across the degraded network go through a Single UDP Pipe.

Typical ROS operation proved problematic for the Trials. In the VRC, teams were allowed a setup period with unconstrained communication before each task began. This makes sense conceptually, since any disaster relief robot will presumably start its activity in a place with good communications. However, in the DRC Trials, network constraints were always active. As described above, when initializing a ROS system, a node contacts the master for each topic (i.e., connection), obtains information about the endpoint node, then establishes a connection to the endpoint. The constant low bandwidth and high latency at the Trials made standard multiple cross-network "handshaking" impractical unless explicitly handled.

The network-specific software architecture shown in Figure 24 extends the one we created for the VRC. Some VRC techniques transferred directly to the Trials: bandwidth reduction via data compression, data type downsizing (e.g., using 32- instead of 64-bit floats, replacing strings with enumeration constants), and other methods described later in this section. Also retained was the message management client/server pair, which provides a single connection between the User and Field computers. The message manager provides a per-topic throttling mechanism by which the user is able to set a periodic transmission rate, "pull" individual messages on request, or turn off data

transmission altogether. Our VRC setup is augmented for the DRC Trials in three ways: (1) multiple masters, (2) the use of a *single UDP pipe* (SUP) pair, and (3) *ping* nodes.

To circumvent the handshaking problem, we configured each computer to run its own master. However, a standard ROS system has no mechanism by which masters can exchange information; establishing local, per-machine masters means that white-pages queries are limited to local, per-machine nodes. The addition of the SUP pair resolved the issue by providing a single channel "bridge" between masters that allows creation of a "proxy" for nodes located on the other machine. The ping nodes, one on each machine that monitor the current network state and relay it to other nodes on that machine, were a final step that added a level of intelligence to message caching. Knowledge of the current network state allowed holding messages that would saturate the connection until adequate network resources existed. While the system self-monitored to make sure network traffic would be successful, the operator was also assured that the freshest data available (that could successfully be transmitted) was received.

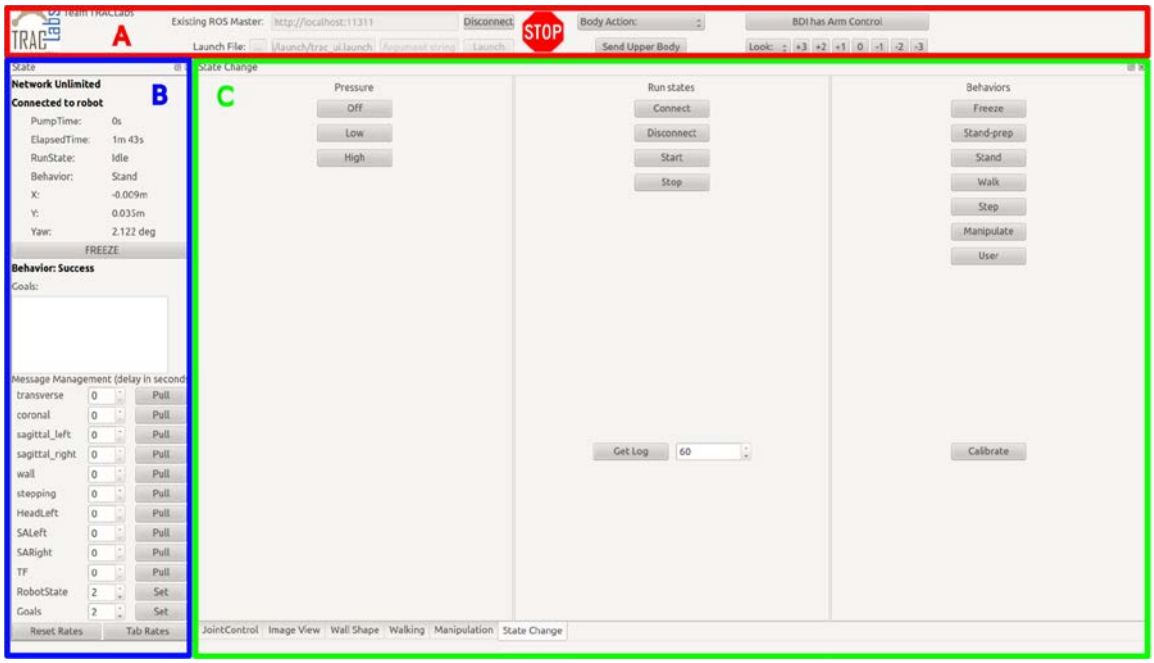

Figure 25: The TRACLabs Operator Interface. (A) controls for connection and always-available robot commands; (B) common network and robot state, command feedback, and message management; and (C) task-oriented interfaces. Regions (A) and (B) are visible at all times, whereas (C) can be changed by selecting a tab.

3.2.1.2 Operator Interface

The OI is the only means of interaction between the user and the (remote) robot. In an environment where severe network constraints sit in opposition to timely situational awareness and robot control, success demands that the OI handle data well on two levels: processing and presentation. Data processing is supplied by a modular, multi-threaded, and extensible Qt-based infrastructure developed at TRACLabs. Its integration with the

networking components assures that network resources are well-utilized so that data can be presented for operator interaction.

Systems views

In the OI's presentation role, a key principle is to show only the information and commands relevant to the current activity, yet have the full complement easily accessible to the operator if and when desired. There is a (small) set of information and commands to which an operator should always have immediate access (e.g., network state, robot "kill" switch), referred to here as the system views, that occupy the areas labeled (A) and (B) in Figure 25.

Area (A) provides both connection controls and access to a set of the most commonly used commands. Clicking "Connect" causes the OI to join the ROS system, after which the "Stop" button and controls to its right become active. The team's experience, both in the VRC and in preparing for the Trials, was that the included commands were used so often, and across so many of the tasks, that they should always be available and accessible to the operator.

Experience dictated a similar lesson concerning the display of state information and basic network control, shown in Area (B). The top section displays the network and robot status, while the middle section shows feedback from commands sent (generally as a percent completed). Unlike the mostly passive sections above it, the bottom section provides the means to control specific data rates using the message manager in terms of messages per second, or single message requests if zero. Since the system views are always visible, the operator always has the option of adjusting data reception to either a periodic rate or reducing network traffic to near nothing and getting only the data required at the time it is needed.

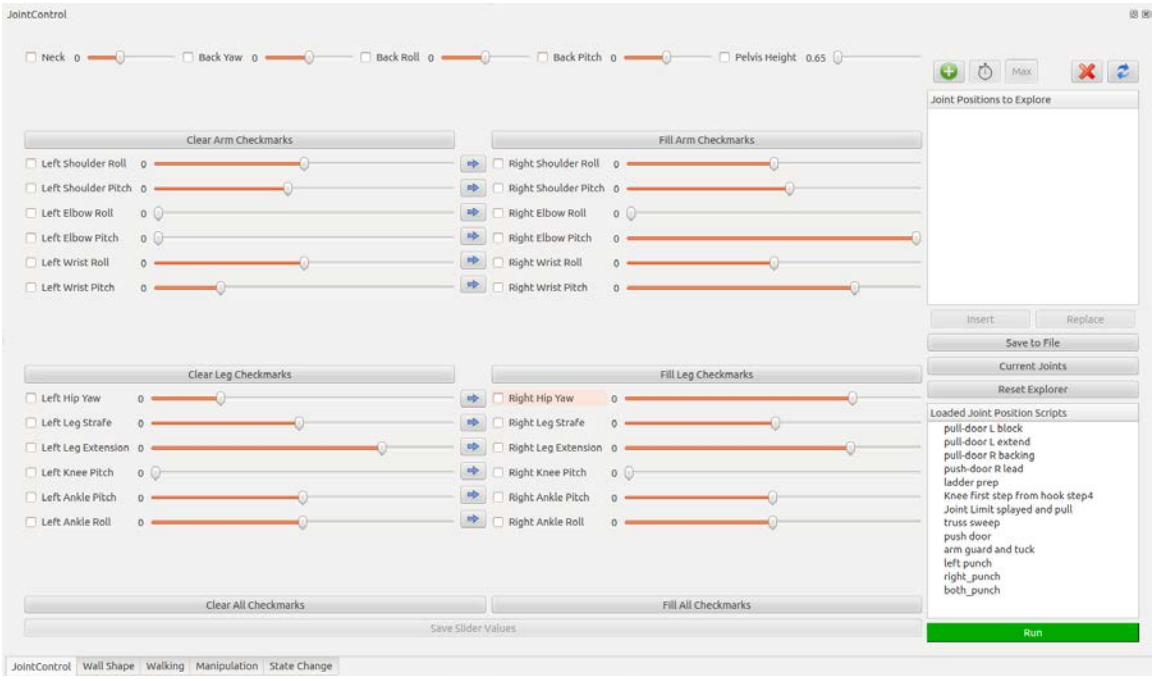

Figure 26: Activity-specific OI tab for low-level joint control.

Activity-specific views

Activity-specific views are selected via one of the tabs along the bottom of the OI and appear in area (C) of Figure 25. Shown is the first of four views discussed in this section: the ROBOT-STATE-CHANGE tab, which accesses the BDI-supplied hardware API. Specifically, the API functionality is divided into three columns and includes turning the hydraulic pump on and off, connecting and disconnecting the on-robot network, and engaging the various behavior controllers provided by BDI. In addition, there are buttons for writing log files to disk for BDI analysis and low-level robot calibration.

The next activity-specific tab is one for low-level JOINT-CONTROL, shown in Figure 26. Joints appear in one of five groups, divided according to robot body-section (one for back/neck and one each for left/right arms and legs). Each individual joint is represented with a selection check-box, description, and slider control. In addition, some convenience controls proved useful: arrow buttons between columns are used to easily transfer settings between left and right joints for symmetry purposes, while clearing or marking all the joint selections in a particular group can be done by a single button click. The right column of the display provides a workspace that enables configuration modification, naming, sequencing (including time delays), saving, and sending to the robot. Values can be set by recalling a configuration or pulled from the current joint values of the robot. Although this type of low-level joint control is time consuming and relatively difficult to use, it is useful in some situations, and many of the commonly used commands were created this way.

The WALKING tab shown in Figure 27 is used for multi-step stepping and walking. Most of the display is occupied by a camera image on the left and a height map on the right. Described in more detail in Section 3.3, it is sufficient here to understand that height maps are a 2D grid representation of the 3D space surrounding the robot. The map for walking is a “top-down” (or X-Y plane) view, where each grid cell is associated with both grayscale and Z-axis height data, obtained from the camera and LIDAR, respectively. The operator can change the grid size and resolution to any one of four settings at will, and toggle between color and height as desired. The controls for selecting the camera source, map size, map coloring, and step-type appear across the top, while stepping parameters (e.g., stride width) appear in a column to the right. Shown in the figure is a path of waypoints for the walk step-type, originating at the robot’s current position (the circle nearest the bottom), specified by clicking on the map. Four other step-types are available, each with a distinctive display indicating the selected step-type: (1) turning (a magenta arrow), (2) side-stepping (yellow “cross-hairs”), (3) back-step (no visual indicator), and (4) double-step (a magenta line whose ends indicate left/right foot placement). In general, only “walking” and “turning” with default parameter values were used frequently, although the other options were readily available to the operator.

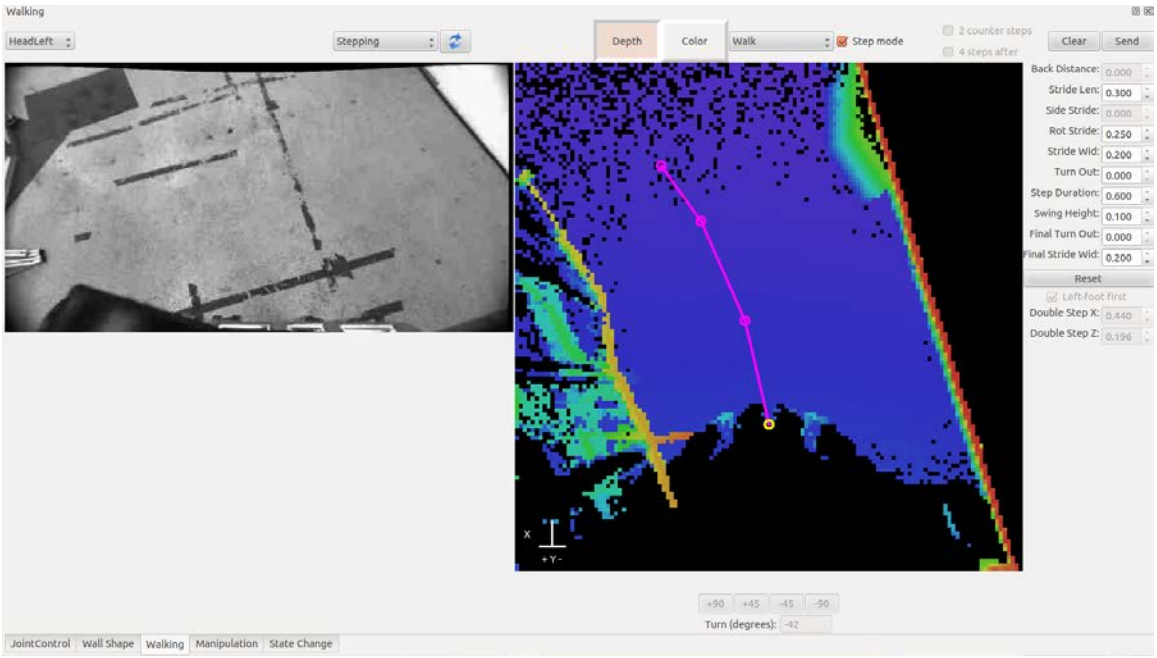

Figure 27: The activity-specific tab for multi-step movement (walking, turning, and side- or back-stepping).

The final activity-specific tab is WALL-SHAPE, developed for specifying a geometric shape on a wall. It has almost the same presentation as the WALK tab, but only the top row of controls is included and the height map displays a "front" (or Y-Z plane) view. Replacing the step-type selection are a button to detect a vertical surface, a toggle for choosing left or right hand, and two data entry boxes for entering "offset" distances. Whereas clicking on the map for the "walking" step-type specifies a sequence of waypoints that define a path, the sequence here defines a trajectory, parallel to the map's plane, for a hand (or tool-tip) to trace. There are two distinct capabilities developed specifically for the WALL task: (1) run-time detection of the vertical surface and its relative pose to the robot, and (2) providing an offset for the distance of both the trajectory from the surface and the initial and final hand locations normal to the surface. The techniques worked well and, in fact, could also be applied to the DOOR task.

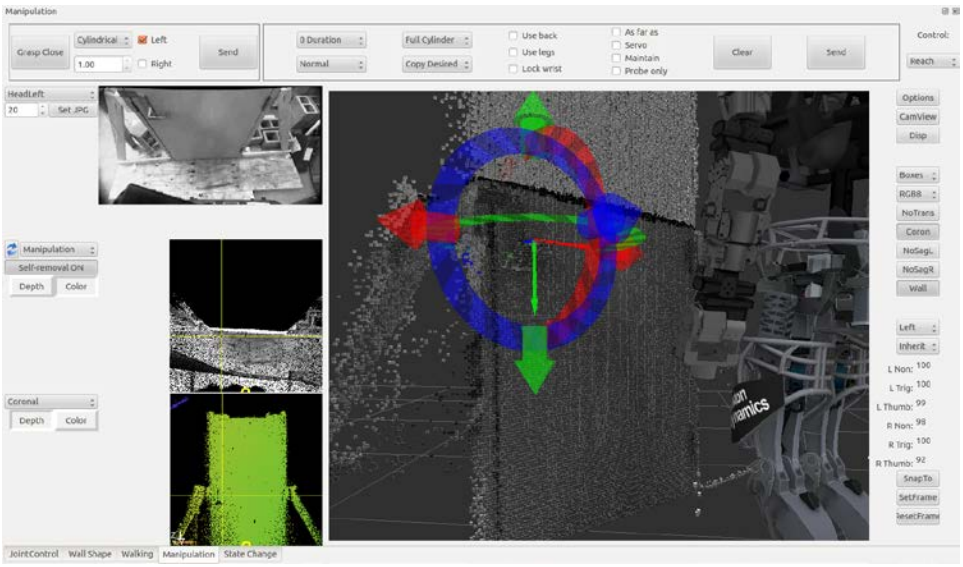

Figure 28: The functional OI tab for working in 3D space.

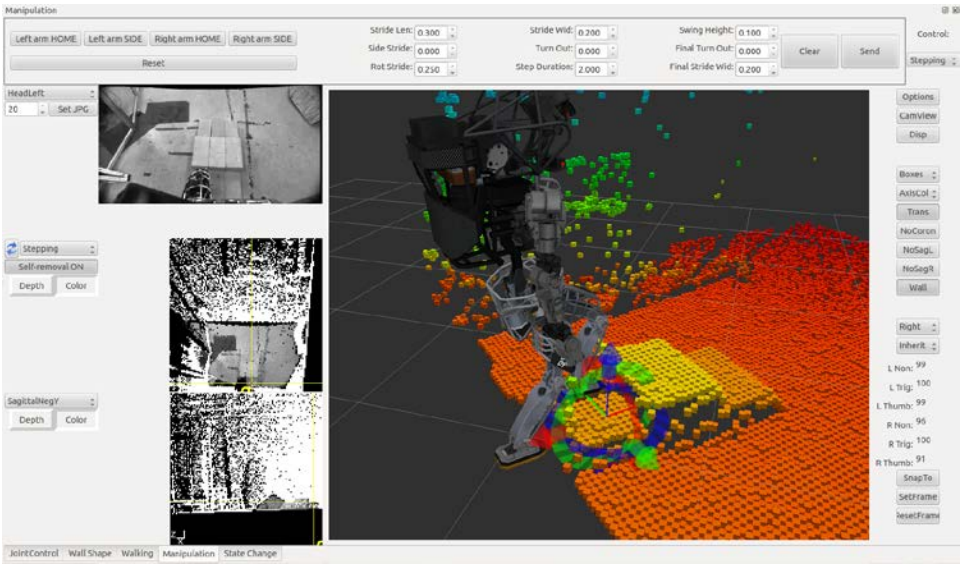

Figure 29: Functional OI tab for stepping.

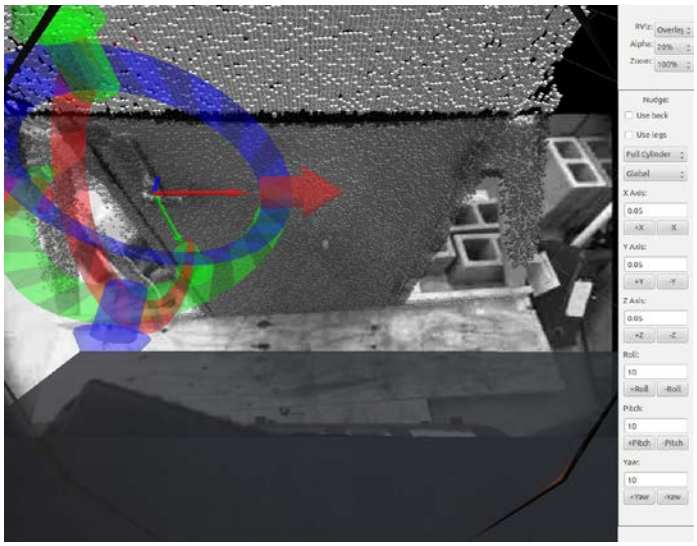

Figure 30: OI window encapsulating the RViz "camera view", which overlays information on a camera image, alongside operator controls for incremental individual Cartesian movements.

Functional views

Functional views also appear in area (C) of Figure 25 and are selected via one of the tabs along the bottom of the OI. The visual structure of the functional view has a camera image and two height maps arranged in a column to the left, a 3D visualization and controls to modify the viewing options to the right, and an activity specific set of controls across the top. The top map (middle of the column) always displays the “top-down” (X-Y plane) viewpoint, while the bottom map can be changed as desired to display any one of the “front” (Y-Z plane) or “side” (X-Z plane, to either left or right) maps. The 3D visualization is a component of ROS’s Qt-based Robot Visualization (RViz) tool, which integrated fairly easily with the other OI components.

Figures 28 and 29 shows the MANIPULATION tab as it is used for opening a door and for precision foot placement, respectively. Whereas the activity-specific WALL-SHAPE and WALKING tabs from the previous section provided a way to specify a (coarse) trajectory, these tabs are used to move to a precise location in 3D space. The coupled relationship of the maps and point cloud reinforce one another: choosing a 2D point from each of the maps (which overlap in space and share a common axis) defines a point in 3D space. This is demonstrated by the “cross-hair” lines drawn on the maps, with the resultant location of the “rings and arrows” interactive marker that appears on the 3D representation.

This tab has some highly desirable capabilities not found in the activity-specific tabs, notably the more precise poses, inclusion of a robot avatar, the ability to “fly around” the 3D world, and a visual method of determining orientation by aligning the planes of the interactive marker with a human-estimated feature in the point cloud.

Finally, Figure 30 shows an extension of the 3D visualization referred to as a CAMERA-VIEW. It shows an image that is fixed from the perspective of one of the head cameras, overlaying the reconstructed point cloud, “rings and arrows” interactive marker,

and other visual data representations from the 3D view. The controls to the left allow an operator to enter individual incremental Cartesian adjustments to the interactive marker. This view is not shown by default, but can be called up with a single button click if the operator requires additional environmental context.

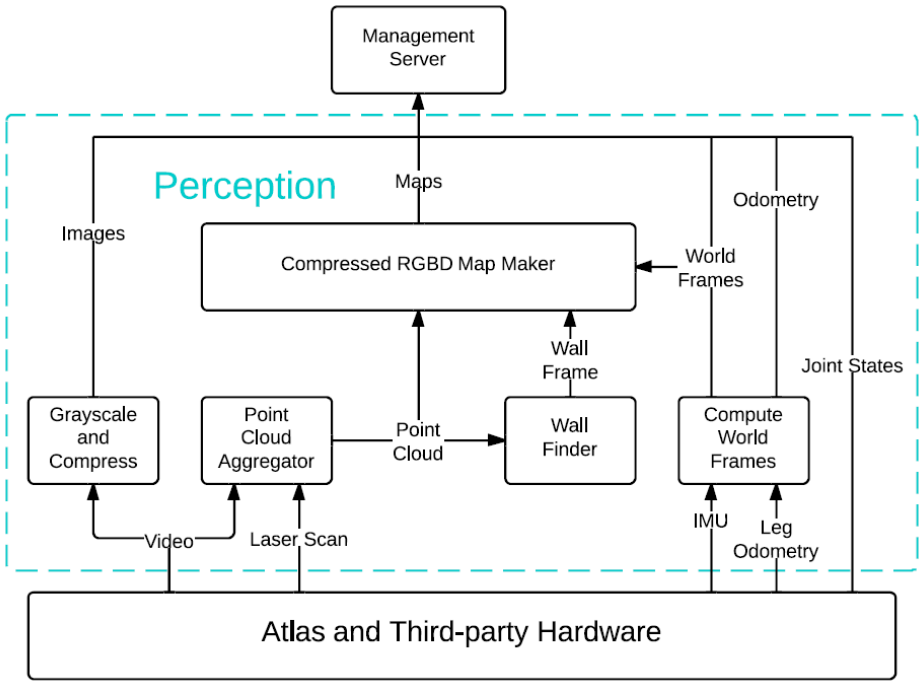

Figure 31: Functional schematic showing the various components used in perception.

3.2.1.3 Perception

The hardware configuration of Atlas presents some unique challenges. Issues with its perceptual capabilities generally fall into two categories: those that are inherent in the given hardware and those that only become issues for remote operation under the imposed network constraints. The following discusses both types of perception issues. Figure 31 shows the block diagram for the perception components.

Hardware-related perceptual issues

The most pernicious issue we experienced has been a lack of adequate real-time 3D perception. While the stereo camera pair in the Carnegie Robotics head can provide a point cloud from 1 megapixel images at 30Hz, not only is the 45-degree FOV too narrow to provide the desired environmental context, but the cloud it produces is noisy and has large gaps. The alternative, which we chose, is to obtain a front-facing 3D point cloud from the spinning LIDAR. In practice, real-time maneuvers present a problem: if the LIDAR spins slowly, it takes a prohibitively long time to get a full sweep of the environment, whereas if it spins quickly, large portions of the environment are skipped

and critical information is missed. Our solution is to maintain a level of point cloud persistence through aggregation over time.

Maintaining perceptual persistence requires accurate odometry. That is, integrating new sensor readings depends intimately on having an accurate record of a reading's location and time. While Atlas' IMU provides a fairly accurate measurement of its absolute 3-DOF orientation, only an estimate of absolute position is available. This estimate is accurate during predictable action, such as taking slow, stable steps, but degrades significantly when performing highly dynamic movements, as happens when using BDI's walk mode. Presumably, this is due to error introduced by events that cannot be accounted for in the algorithm (e.g., foot slippage, temporary loss of ground contact, etc.).

A third issue we encountered was an inability to focus the sensors on close-in manipulation due to a confluence of sensor coverage and joint kinematics issues. The sensor head is mounted on a single joint attached to the top of the torso and can only pitch up and down; from the waist up, the robot has no side-to-side rotation joints, such that the center of the chest is always aligned horizontally with the sensor head. While sometimes awkward, this setup might be adequate for manipulation–except that the kinematic restrictions of the arms often makes it physically impossible to both "look at" and manipulate an object at the same time. This problem was recognized during the VRC, after which the (fisheye) SA cameras were added to Atlas. While significantly helping human operators, the resultant SA images are difficult to use for two reasons. On the one hand, the high distortion near the image edges (due to the fisheye lenses) is so great that standard computer vision techniques fail. On the other hand, although the relatively small distortion near the image center could be inverted, the cameras are mounted rigidly to the robot body, and cannot generally be pointed directly at an object being manipulated.

To overcome most of these problems, we simply reduced movement speed. Doing so resulted in good-enough proprioception-based position estimation, allowing a simple aggregation of laser scans into a cohesive 3D model of the world. Furthermore, slow movement had the added benefit that it reduced effects of sensor lag. Also, in cases where the environment to be manipulated was out of the FOV of the head cameras, the operator used the point cloud and SA cameras to manually estimate the location of objects.

Network-related perceptual issues

By necessity, communication between the User and Field computers had to be severely limited. The fact that our general approach to the competition was essentially human-in-the-loop magnified the impact. In practice, reducing the data flow amounted to the reduction of two data types: camera images and point clouds. While other data types were a concern (e.g., joint states, command feedback, force sensors), their combined bandwidth is dwarfed by that required for images and point clouds. A brief discussion of reducing camera images follows, while the remainder of the section details the method applied to handling point clouds.

The high-resolution images produced by the various cameras individually require more bandwidth than is available. The obvious solution is to greyscale, downsample, and compress them to a manageable size, even though there is a significant impact on image

quality. However, our reliance on human interpretation makes it imperative that images be of high enough quality for the task at hand. By default, images are simply not sent over the network until requested and, when they are, apply a very high compression level. Extending the team's approach of human-reliance, both the image rate and its compression level can be changed by the operator during task execution as desired.

A full 3D point cloud, potentially consisting of millions of points, simply cannot be transferred over the network under competition conditions. To provide an accurate representation of the environment with a small enough data size, the continuous 3D point cloud is "flattened" into one or more 2D, discretized projections, or "height maps". Given the spatial frame definition in which the X-axis points straight out in front of the robot, the Y-axis corresponds to the left and right sides, and the Z-axis represents height, projections on the X-Y, Y-Z, and X-Z planes yield "top-down" (transverse), "forward" (coronal), and "side" (sagittal) views, respectively. A fixed-size grid, with cell sizes of some constant resolution, is overlaid on the continuous-valued projection, grouping the points in each 2D grid cell. Points in each cell are analyzed to find the minimal or (maximal) value along the third dimension, which provides both a height and color for the cell. Prior to transmission, a compression algorithm is applied to the result to further minimize data size.

Once received, a height map can be displayed directly as an image (e.g., Figure 27) or unpacked to reconstruct a colored 3D point cloud (e.g., Figures 28 and 29). Since the grid size and resolution of a specific map may not be suitable for a particular task, several pre-configured map sizes have been defined that have proven sufficient for all DRC tasks (ranging from a coarse 128x128 grid with a 0.8m resolution to a highly detailed 240x240 grid with an 8mm resolution that can be selected by the operator as desired). Empirical results indicate that transfer of the full point cloud would require messages that averaged around 600KB in size, while the height maps generally averaged around 10KB each. While all of the transverse, coronal, and sagittal maps are available, any or all could be turned on or off at will, and the message rate of each could be set independently. Practically speaking, the sagittal maps were used only sparingly if at all; for many tasks, only the transverse (e.g., walking or stepping) or coronal (e.g., wall cutting) maps were required, providing a huge reduction in the bandwidth used.

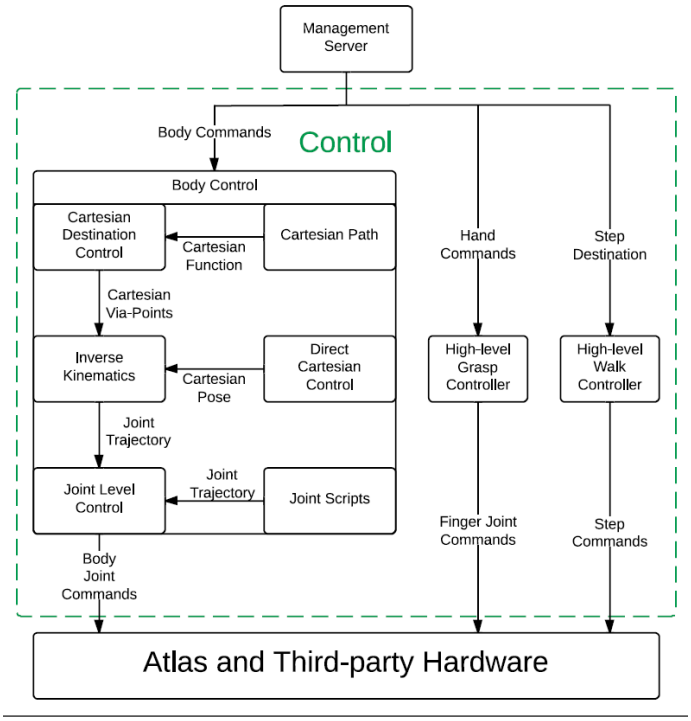

Figure 32: Schematic diagram showing the Control module from Figure 23.

3.2.1.4 Control

The overall control architecture is shown in Figure 32.

Mobility/walking

A robust walking controller for a humanoid robot is very challenging to develop, but is clearly necessary in order to be able to execute tasks successfully with Atlas. One of the advantages the Track B & C teams had over Track A teams was that we did not need to develop our own walking controller, since BDI provided a robust one. In a way, this was also a disadvantage, as we could not examine the software or algorithms, nor make any low-level changes to the controller. It is essentially a “black box”. For Team TRACLabs, the advantages outweighed the disadvantages, and we used the BDI controllers very effectively to accomplish the Trials tasks. Other Track B teams with more experience with balancing, walking, and whole-body control looked at using their own walking and balancing controllers.

BDI provides two controllers for mobility:

(1) Quasi-static Stepping Mode: In this mode, Atlas transfers its Center of Mass (CoM) over the footprint of one foot in such a way that it can balance on that foot while lifting the other off the ground. It then lifts the other foot and places it at a new location on the ground, and then finally shifts its CoM back to a more balanced location between the two feet. This mode is useful when the desired foot placements are on a surface that is

not horizontal, such as when traversing over the ramp or randomly oriented flat surfaces in the Terrain task. At any moment, robot motion could freeze and the robot would simply balance exactly where it is. Unfortunately, the robot must sway sideways quite a lot to get the CoM over each foot, and it may collide with the environment around it if obstacles are nearby. Thus, for instance, it is impractical to use this mode to walk through a doorway.

(2) Dynamic Motion Walking Mode: Here, the Atlas transitions dynamically between steps in the standard falling inverted pendulum motion of human walking. This mode requires several pre-planned steps ahead of time, and needs to know which is the final step, so it can halt the momentum of the body. It is fairly robust to external environmental disturbances, such as brushing against a doorway while walking through it, but is definitely not quasi-static: if the robot were to freeze in the middle of executing this motion, it would fall.

We created several higher-level walking and stepping modes that make use of BDI's controllers. These are rotate-and-move, only rotate, only move to the side or backwards, and two-step (directly forward or backwards), and are available as options on the WALKING tab, shown in Figure 27. Moreover, the operator may specify various parameters associated with the steps, such as stride length, swing height, and step duration, by entering the desired values in the OI. In the case of flat, even terrain, the operator can specify a destination point in the map rendered in the OI and our foot placement planner, based on linear interpolation, would determine the steps. The Atlas would then execute the steps autonomously. This is a good example of the semi-autonomous control mentioned in the DRC objectives. For more complicated tasks, such as walking over uneven terrain, the operator specifies the target pose for the sole of the foot for each step using interactive markers, as shown in Figure 29.

Manipulation

In both the VRC and the Trials, manipulation required trajectory planning, which in turn requires a fast and robust inverse kinematics (IK) solver. During the VRC, we found that the third-party Kinematics and Dynamics Library (KDL) [Bruyninckx and Soetens, 2002] did not perform well. It often failed to find solutions for poses that were actually reachable, and rarely could deal with singularities in joint space. This made it nearly impossible to generate long trajectories and approximate Cartesian motion, because the probability was very low that solutions would be found all along a trajectory.

Furthermore, when solutions were found, it took longer than we liked to find them. These issues are primarily caused by the fact that the KDL solver's performance is highly dependent on the seed value joint configuration. If the seed configuration is close to the real solution, the solver typically converges to the real solution much faster than with a random seed. In an attempt to find better seeds, and thereby make KDL usable, we developed a lookup-table (LUT) method for finding joint configurations that would place the end effector near where we wanted it. This LUT method worked well enough for the VRC.

In our LUT implementation, millions of forward kinematic solutions are recomputed offline, creating a map from Cartesian space to joint space. These joint space/Cartesian space pairs are stored in a database, which is loaded into memory at run time. Databases

are created for four kinematic chains: one each from the pelvis to the left and right hands, and one each from the upper torso to the left and right hands. The user selects the appropriate database to search when creating a trajectory. The databases are parsed into kd-trees to support a Cartesian-space search that returns the 100 nearest neighbors to serve as joint configuration seeds for KDL. Although finding a reasonable solution was much more likely with a selection of seeds known to be near the pose, the IK still failed too often. To further decrease failures, an option to ignore the orientation error was added that would instead choose the solution only considering the closest position.

While preparing for the DRC Trials, we tried out another third-party solver (IKFast [Diankov, 2008]) as well as the precomputed LUT system we developed for the VRC. Neither worked well enough to perform the tasks in the Trials. We found that they illustrated several problems with rigid IK solvers in general.

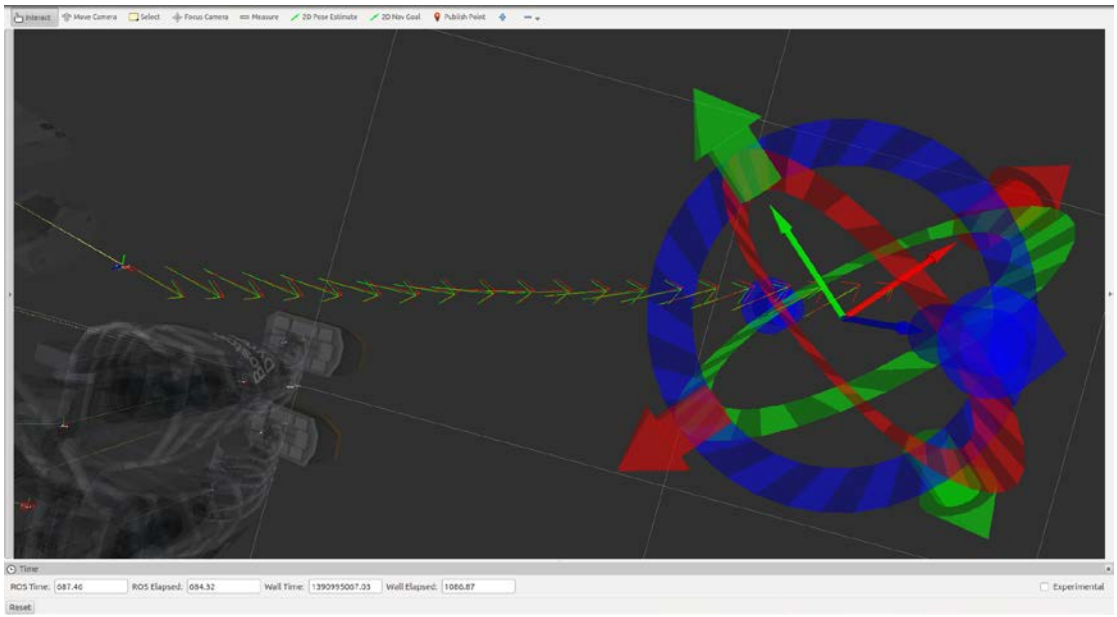

Figure 33: Example of a trajectory (red arrows) and solution (lighter, green arrows) generated by our algorithm. Notice how closely the green arrows match the red arrows.

Many robotic manipulators, including Atlas, are unable to reach some subset of 6-DOF poses within their workspace. These pockets of unreachable space make it difficult to plot a Cartesian path between two poses, because it is not obvious which poses along the way will be unattainable. Typically, however, strict Cartesian motion with high precision is not required. For instance, grasping a hose a few millimeters higher than expected, or at a slightly different angle, is often sufficient for the desired outcome. An "error tolerance", potentially task-dependent, is therefore useful. Unfortunately, KDL and IKFast do not support such functionality in a controlled manner (you can increase an overall epsilon error value, but can give no direction as to where that epsilon may be applied).

In addition to the problems caused by insufficient configuration space is one of unpredictability. Even with perfect IK solutions, a robot may not interact with the real

world the same way every time it performs a task — tools and other objects will be grasped in different ways and may shift during use. This motivates the concept of controlling a dynamic tool frame offset that can be set after a grasp is made, and can shift over time to match reality. This would be very difficult to implement in IKFast or our previous LUT scheme, since it would require numerous copies of the solver/LUT for all possible perturbations of the tool/object.

Most IK solvers also have very little flexibility in terms of the kinematic chains that can be used. This is a problem because there are often situations where you only want to move a subset of your joints. For instance, when moving a single arm, most of the joints are available to the solver, including leg and back joints. However, when performing a dual arm manipulation, the leg and back joints must support the combined effort of both arms. We also often found it useful to lock two of the back joints during most manipulation actions to make it easier for Atlas to keep his balance. IKFast and our LUT method require offline computation for each chain to be used. IKFast also requires the number of unknown joints to match the selected IK mode, forcing the operator to manually set/search the remaining joints if there are more than the solver is prepared to handle. KDL has trouble finding solutions when you try to force a joint to remain constant.

To solve these issues, we developed a flexible simulated annealing algorithm similar to [Dutra et al., 2008], which optimizes joint positions using a cost function based on the quality of the forward kinematic solution. However, our algorithm differs in five ways:

1. To find a solution at any via-point, we use the solution to the previous via-point as the starting point of the search. Since our trajectories consist of a large number of via-points interpolated in Cartesian space, each via-point is very close to the next. This means that the distance in Cartesian space that the solver must move is small, and therefore the distance in joint space is likely to also be small, often resulting in significantly faster solution times.
2. We limit the search space for the simulated annealing algorithm to only the range of joint positions that can be reached within the time allotted for the movement while traveling at less than the maximum velocity for the joint. Decreasing the search space increases search speed, especially for long kinematic chains.
3. We add a “movement factor” term, which allows the user to favor large movements in some joints over others, or even disallow movement in a joint completely.
4. We incorporate user-defined tolerances for all six Cartesian degrees of freedom into our cost function, which allows the solver to accept imperfect solutions in the event that an exact solution does not exist within the robots configuration space. This is frequently the case with Atlas.
5. We add filtering to the trajectory as a whole by first “tightening” each solution toward the one before it (minimizing the distance in joint space between subsequent poses), and then low pass filtering each joint’s full trajectory individually to reduce high frequency noise.

Since this method incorporates error tolerance into the solver’s cost function, a wide range of tolerance options can be implemented. It only needs the forward kinematic solution for the arm, making it trivial to add and modify tool frames as necessary. For the same reason, it also has no problem handling complex chains or locked joint positions.

To supplement the IK solver, two different Cartesian space via-point generators accept options of straight-line Cartesian motion and arced Cartesian motion. The user also has the option of real-time Cartesian space control by means of continuously moving a 6-DOF Cartesian waypoint. At the lowest level, the body control system accepts direct joint commands, as well as enumerations to preset joint-level trajectories. The hands are controlled by a tight loop using only sensors in the hands themselves to allow for an independent system capable of several simple grasps that conform to objects in the hands. Refer to Figure 32 for a pictorial representation and Figure 33 for a visual example.

3.2.2 Vehicle task

We decided not to pursue the Vehicle task fairly early in the four-month development period. Initially, the robots were expected to enter the vehicle before driving, and we felt that we did not have the resources to solve that problem. We would have needed to purchase a Polaris to be able to develop the ingress and egress scripts, and our experience on the VRC made us wonder whether we could do it even then.

3.2.3 Terrain task

Difficulties to overcome in this task include the robot's kinematic limits, perceptual limits, and OI capabilities. Kinematic limit issues are encountered because it is physically impossible to perform certain movements (e.g., step from one skewed platform to another in certain configurations).

For autonomous stepping, a set of foot placements is generated using the normals of perceived step-surface planes. Experience dictated that this failed too often to rely on as a competitive strategy, due to both perception and mobility issues. Instead, we chose to utilize the OI capabilities, relying on the operator's ability to choose foot placements. For each step, the operator matched a pose marker with their interpretation of the point cloud (see Figure 29). The robot then placed the foot at the commanded location with the commanded pose.

We used the iRobot hands for this task. We did not use them for manipulation, but rather they were the heaviest hands in our bag, and we wanted to lower the center of mass as much as possible.

3.2.4 Ladder task

The primary difficulty to overcome proved to be the robot's mechanical and kinematic limits, although the mechanical limits of the hands also contributed. Climbing the ladder facing away from the ladder was impossible because the tether collided with the ladder. Climbing facing toward the ladder using BDI's stepping/balancing controller was also impossible, because raising a foot during the step motion caused the knee to collide with the next step on the ladder. Neither the iRobot nor Sandia hands could adequately support the forces required when grasping the railing. This left the BDI hooks as the best available option, but these could not practically be used on the handrails, so needed to hook around the steps of the ladder itself. Together, these limitations implied that a scripted full-body motion, and possibly custom hands, would be needed in order to

ascend the ladder. It was apparent that other Atlas teams experienced the same issues; teams that scored points all seemed to use replacement hands or custom hooks of their own design and full-body joint-level scripting.

Unfortunately, we received the hook hands less than three weeks before we shipped Atlas to the Trials, and were not able to script a successful step-climbing behavior in time. As a "fallback" option, we decided on an unorthodox method to score one point. A joint-level script was created in which the BDI hooks were engaged on the fourth step; the support provided was enough to keep Atlas stable while sliding the knees over the first step, resulting in a kneeling position with feet raised off the ground. This pose was sufficient for the first point, but no further climbing was possible.

We used the BDI hooks for this task, in order to attach to the fourth step.

3.2.5 Debris task

Difficulties to overcome for this task included kinematic, mechanical, and perceptual limits. The conventional method of removing obstacles piece-by-piece was subject to all the difficulties in various ways. Simply getting Atlas into a position to perceive specific objects for manipulation was difficult due to the perceptual issues outlined above. In addition, both the iRobot and Sandia hands were too weak to securely grip and hold an object unless grasped near its center of mass. Available arm configurations severely limited reachable locations. This was further exacerbated when accounting for the orientations necessary for grasp points. Furthermore, the arms did not reach the ground with Atlas in a squatting position: pitching the back joint beyond a certain point threw Atlas off-balance. Other Atlas teams all used custom manipulation devices that extended hand reach and improved hand strength.

Although we experimented with several custom hand options, we found that we could not reliably grasp the debris. Instead, we devised a different technique for debris clearing that we call "plowing". Experiments proved that BDI's dynamic motion walking controller was quite resilient–short, fast, shuffle-steps served to push the truss and debris out of the marked area. This motion generally led to the loss of balance and a fall, but often succeeded in moving a lot of the debris first.

We used the BDI hooks for this task. Although our strategy did not require manipulation, these were the longest end effectors we had in case we needed to try to reach a piece of debris.

3.2.6 Door task

Difficulties to overcome for this task included robot control/stability, perceptual limits, and OI capabilities. Simply locating the latch, positioning the hand, and identifying when the door was unlatched involved the perception and OI. However, the overwhelming challenge turned out to be maintaining robot stability when making contact with a solid object: pushing or pulling in opposition to an obstacle affects robot balance, with too much force resulting in a fall.

The stability issue is clearly demonstrated in both door opening (including turning the latch and opening the door) and threshold crossing. When turning a latch, avoiding unbalancing forces requires a hand trajectory that moves with the latch's plane of

rotation. When opening a door, it requires a trajectory in a direction in which the door "gives". In either case, force exerted against a solid object by misdirected motion causes the robot to fall. The issue occurs in threshold crossing also, usually as a matter of colliding with the solid door frame. Assuming use of one of the BDI locomotion controllers outlined above, the robot can either side-step or walk straight across the threshold. At approximately 32 inches, Atlas's shoulder-to-shoulder width is greater than its depth from front to back, making collisions less likely with side-stepping.

However, in addition to taking considerably longer than walking, side-stepping causes a lot of side-to-side robot "sway"; the line of robot travel must be very well aligned to avoid contact at the furthest leaning angles. Other Atlas teams typically used either the iRobot or Sandia hands to grasp the door latch, then chose side-stepping to cross the door threshold.

Rather than use any of the DARPA supplied hands, we used the mounting plates as our arm end effectors. The inspiration for this stemmed from the frustration of consistently snapping iRobot finger tendons when experimenting with door-opening techniques. We discovered that the minor lip of the plates provided a recess that the door handles fit into nicely, while their circular shape allowed them to slide down the length of the latch after turning. The plates worked better than the BDI hooks, which were difficult to position in a way that gained purchase on the latch and were difficult to remove once "hooked". We used the trajectory method of arm motion explained above to turn the latch, fine-tuned if necessary using the manipulation method also described above. Scripted motions served to open the door once unlatched, and to put the arms in a configuration that minimized the shoulder width before using dynamic walking to cross the threshold. We also encountered an environment issue–a breeze blew the second door shut after we unlatched and started to open it.

We used no hands for this task, relying on the bare mounting plates instead.

3.2.7 Wall task

Both robot control and kinematic limits presented difficulty, with hand control presenting a particularly tough problem. Pushing a rotary tool sideways through a wall exerts considerable forces and torques. Without a relatively constant tool-tip position relative to the wrist, the drill would shift and the resulting cut would drift outside the lines. Maintaining the tool-tip position requires a very strong grip; the default hand-force limits were not strong enough, yet the likelihood of snapping tendons increased dramatically when they were increased. Besides grip issues, finding correct kinematic solutions for cut trajectories was difficult. Some other Atlas teams used custom hands to resolve the grip issues.

We chose to use the "gun-style" drill, as it was both easier to switch on and had a second handle for stability. We also selected the iRobot over the Sandia hands, both for their stronger grip and easier repair. While we experimented with a dual-arm strategy, it proved too unreliable and too often resulted in hand damage for actual competition.

We mounted two iRobot hands for this task, although we only used the left one during execution. The second hand was there in case we broke a tendon on the first.

3.2.8 Valve task

Difficulties encountered include robot stability, kinematic limits, and UI capabilities. As with the Wall task, robot stability was affected by solid-body contact, while the kinematic limits made it difficult to find adequate arm trajectories. The effect on robot stability was especially noticeable on the 8" handwheel, for which friction increased significantly as the valve reached its closed state and required more force to turn. To address kinematic limits, we identified two particular techniques for valve turning: (1) specifying hand positions, either one-by-one or as a sequence, and (2) calculating an arc of the circle with the same radius as the valve, centered on the valve's axis to describe end-effector motion. Both techniques tested the OI capabilities, especially when the hands blocked perception. Other teams used custom rod-like end-effectors that not only reduced perception blockage, but were easier to insert between valve spokes than the available end-effectors.

Partly to avoid hand damage and partly to reduce perceptual blockage, we used the "nubs" supplied with Atlas as end-effectors. (These ordinarily serve to protect the wrist force/torque sensors.) Their rounded rubber construction provided protection, had a low profile, and had a high degree of friction. We used both valve-turning techniques, chosen by the operator to match the situation.

We used two nubs in place of hands, and used both during execution.

3.2.9 Hose task

Difficulties included robot stability, perceptual limits, robot control, and OI capabilities. Maintaining robot stability was essentially making sure the robot did not trip over the hose when walking to the wye. The latter three issues all came into play to varying degrees when manipulating the hose.

We approached both the walking and hose-handling as exemplars of robot locomotion and fine-grained manipulation, respectively. Positioning the robot used the multi-step UI described, while handling the hose relied on the manipulation OI.

We used the iRobot hands for this task.

3.3 DRC Finals

3.3.1 Infrastructure

1. It became apparent early in the time between DRC Trials and Finals that DARPA management really wanted the teams to attempt the Vehicle task. Thus, we purchased a Polaris vehicle to investigate that task. The vehicle was modified to have a remote kill switch and to have six e-stop buttons mounted on all corners.

2. We doubled the size of our Atlas lab to accommodate the Polaris, greater mobility for Atlas, and stringing tasks together.

3. We purchased a lightweight Aluminum gantry and a 480V 3-phase electric generator for taking Atlas outdoors.

4. The robot was shipped back to BDI in November, 2014, and we had a fully functional robot back in our lab by the end of March, 2015. From then until the Finals, it was plagued with failures that each required downtime until BDI could send a tech to fix it. The faulty valve seals were particularly troubling, since BDI clearly knew of the problem, and that there were many seals on the robot destined to fail. Nevertheless, they only replaced the one that failed the first two times they came out. Finally, they offered to replace the whole set less than two weeks before the competition. We assume there was an issue with supply, but this was still frustrating.

3.3.2 Collaborations

We worked closely with IHMC to try to integrate their full-body control into our software architecture. Just before the robot was shipped back to BDI, we were able to run their code on our computers and control our robot through our OI using it. Unfortunately, our strategy relied on using BDI's mode API, and IHMC's code did not have a way for us to use that functionality. This was partly due to BDI not exposing the necessary components to IHMC, but the impression we got was that neither party was truly interested in solving this problem. In the end, we had to abandon this line of development.

We also worked with Maurice Fallon of MIT, to integrate their 3D localization code. Although this line of development also looked promising, it too ultimately failed. It seems that sophisticated software tends to have such brittle and precise requirements for the code and even hardware that surrounds it that it is extremely difficult to integrate two such systems together in a meaningful way without gutting one of them.

3.3.3 Software modifications – dead ends

We tried many software paths that were ultimately not used in the Finals.

3.3.3.1 Adaptive control

We explored using adaptive control to solve our modeling issues and facilitate coping with future hardware changes. We also hoped that this technique would enable more robust contact control, due to its ability to absorb disturbances. Although we had some success with steady-state regulation, we were not able to generalize this method to trajectory tracking, and eventually decided it was not appropriate for the given hardware.

3.3.3.2 Autonomous walking over rough terrain

We developed perception techniques that enabled the robot to choose its footfall placements autonomously. This work makes heavy use of the Point Cloud Library (PCL) for point cloud filtering and plane segmentation.

The general idea is to make an occupancy grid for every plane found, rejecting planes that are tilted too far from horizontal to be good stepping surfaces. Thus, if you had a checkerboard of cinderblocks, you would get two occupancy planes: one at the height of the tops of the cinderblocks, and one at the lower level. Figures 34 and 35 show the state of this work.

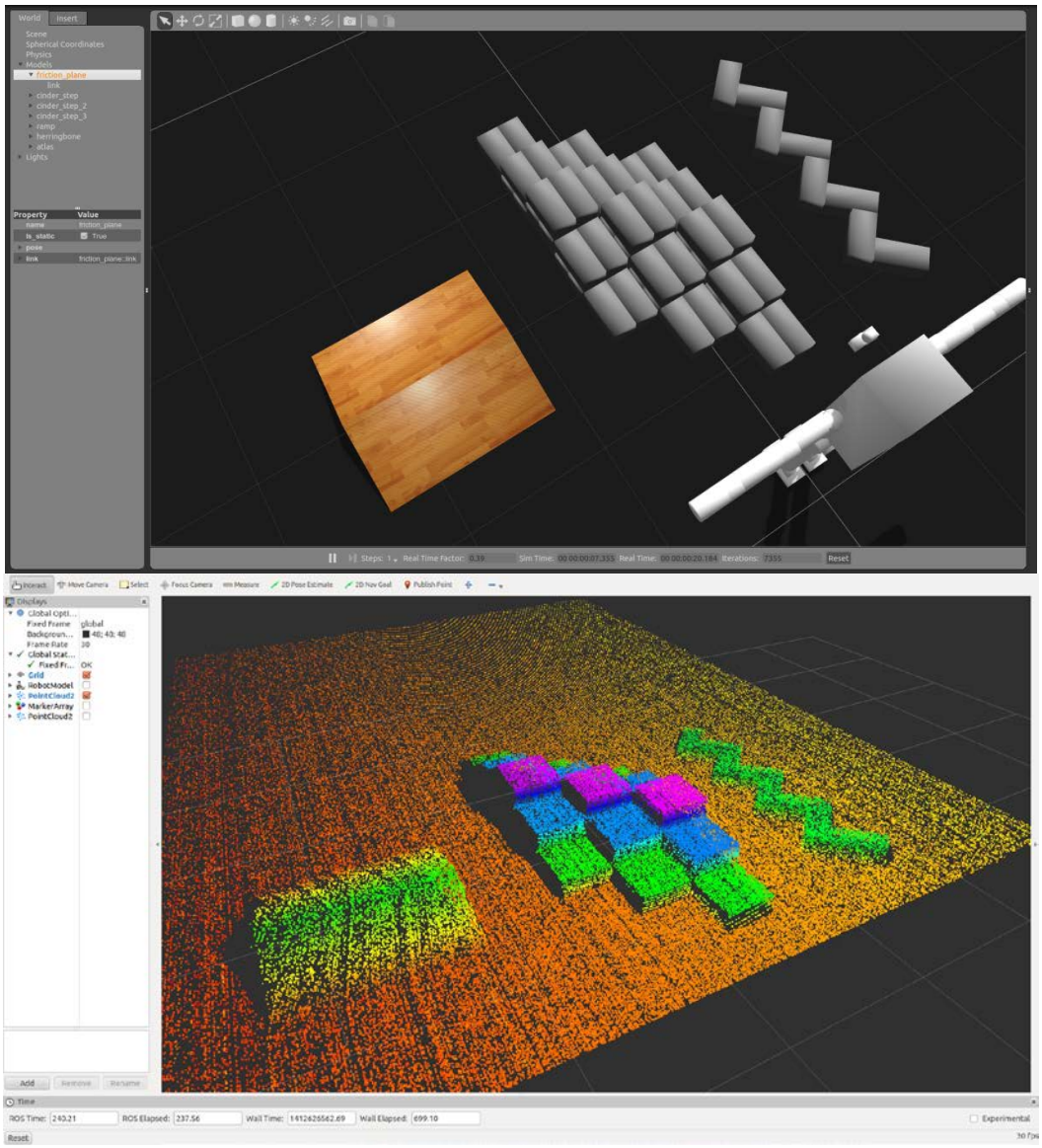

Figure 34: (top) Simulated terrain environment; (bottom) with simulated LIDAR scan.

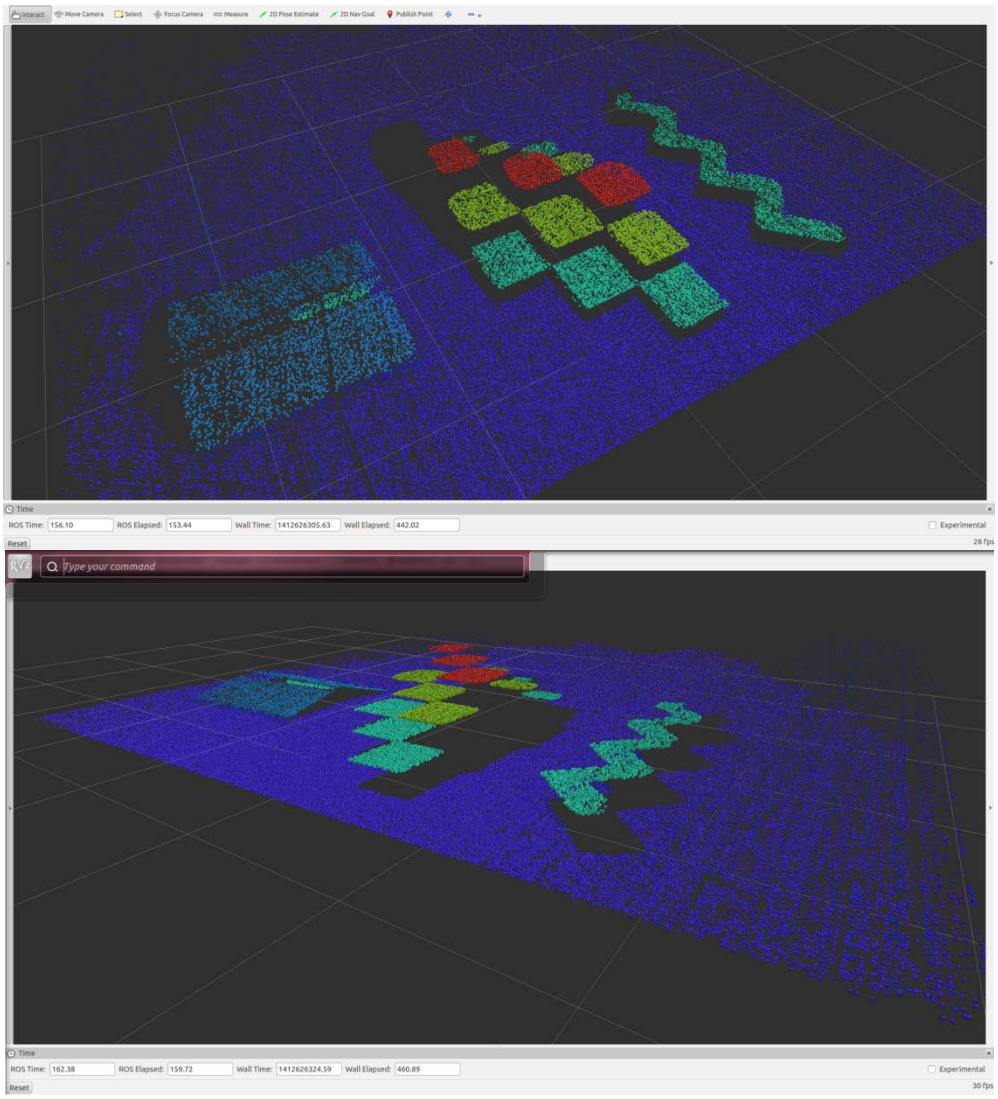

Figure 35: Two views of the planes that have been found in this environment. Different colors indicated the different planes identified.

3.3.3.3 Navigation

The BDI-supplied odometry for Atlas works reasonably well over short distances when we are using the BDI-supplied quasi-static stepping behavior, but is not very robust for more dynamic motions, such as their dynamic walking. We tried implementing visual odometry and SLAM for Atlas using the LIDAR and stereo vision from the Carnegie Robotics Head, in order to improve the quality of robot odometry. As a byproduct, we hoped to be able to use these same algorithms to provide navigation Odometry for the vehicle motions when the robot is driving.

Visual Odometry

The YouTube link below points to a video that shows the results of visual Odometry to accumulate the sensor scan data as the vehicle is driven around the building that

houses our lab. After a drive of several hundred meters, the final orientation is off by perhaps ten degrees, but the location is very good relative to the starting location.

https://www.youtube.com/watch?v=zc7L_yUIylU

SLAM (Simultaneous Localization and Mapping)

6-DOF SLAM in an arbitrary, unstructured environment is very difficult without high end sensing equipment, such as those produced by Velodyne. Even in such cases, the approaches that work best are restricted to the 2D plane (3DOF: X,Y,Theta). In the case of a high degree of freedom bipedal robot, commonly used algorithms for SLAM break down under continuous orientation and height changes of the sensor frame. Using as many as possible of the available sensors on the Atlas robot, we devised a scheme for registering the stereo clouds and laser scans with each other, and localizing ourselves with the cloud. See Figure 36.

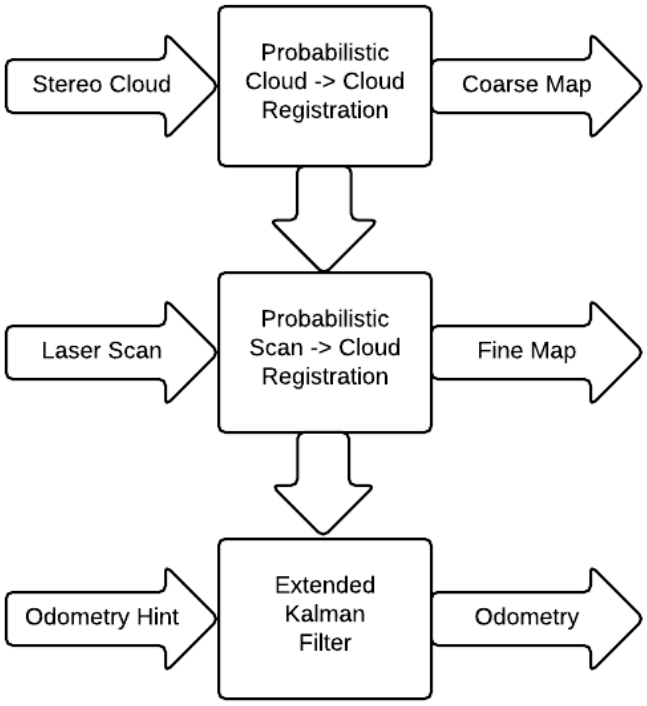

Figure 36: SLAM architecture

The general idea is to probabilistically match stereo cloud to stereo cloud across time by means of a gaussian particle filter around each of the six degrees of freedom. Checks for error are heuristically biased, but are fundamentally simple least squares errors. A similar approach is used to match laser scans to the now-aggregated stereo cloud. The results of each particle filter instance are fed into a Kalman Filter which can be queried at any point for the 'best guess' current odometry. The system can also go back in time and quickly recheck past registrations as new data comes in and is able to go back and correct for past errors. The goal here is to optimize for correctness, rather than performance. This

is still a work in progress. A high-fidelity SLAM scheme such as this will solve many problems throughout our system, and is a key capability for long-term autonomous procedure execution.

Once we determined that the Operator would be able to handle driving without visual odometry or SLAM, these promising projects were abandoned for other more pressing concerns.

3.3.3.4 Gesture recognition

We explored several ways for the operator to interact with the computer on two levels: (1) We want the operator to be able to manipulate objects in the 3D virtual world in an intuitive manner; (2) The operator needs to be able to send administrative commands: changing interface modes, choosing data inputs, sending commands, etc.

For both of these, we used the Kinect to sense the joint values of the operator. The Kinect uses generic skeletal kinematic constraints to determine where key parts of the arms and body are located.

Manipulation

We will track both hands of the operator, and how they relate to the shoulders. These two poses will enable the operator to use two-handed grasps to manipulate 3D targets and objects in the virtual world.

Gestures

We have developed a novel method of parsing motions into gestures, classifying them, and associating those gestures with different desired actions. Currently, we are identifying pointing, waving, and swiping, but this can be extended to other gestures. This work is in its early stages, but shows a lot of promise. Figure 37 illustrates a skeleton model and the ego-sphere representation of a wave gesture.

Ultimately, we did not use this technology in the Finals because we were able to accomplish what we needed without it, and we never fully solved the problem of the immersed operator not being able to use the keyboard and mouse.

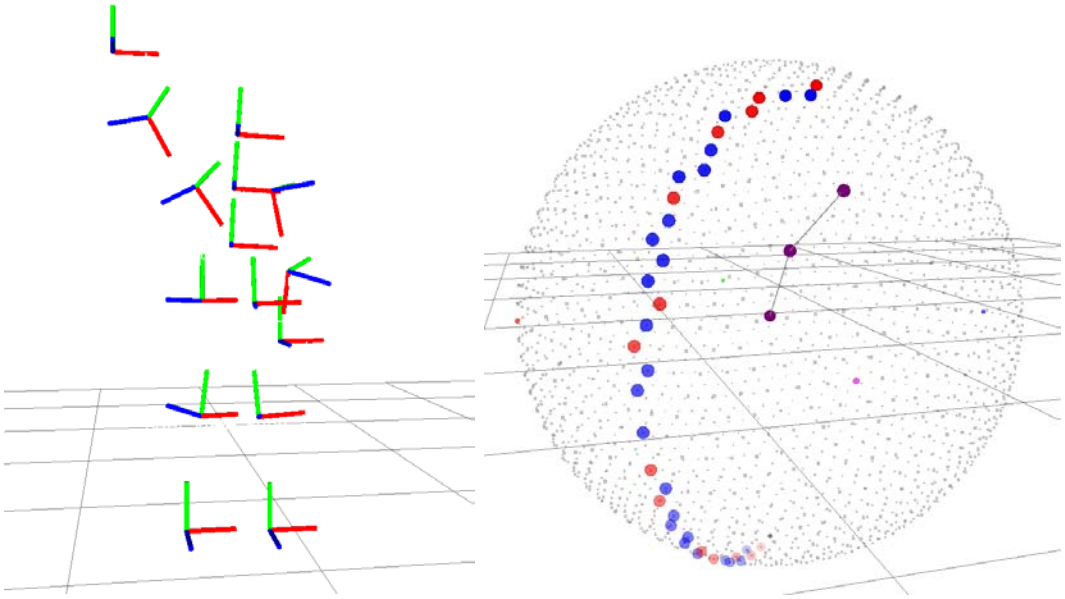

Figure 37: (Left) An example of the skeleton model produced by a Kinect sensor and Openni skeleton tracking software during a wave gesture. The model is composed of 15 skeleton markers, located at the corner of a 3D axis representation: a head, neck, a and torso, and two shoulders, elbows, hands, hips, knees, and feet. (Right): A representation of the beginning of a wave gesture, encapsulated in its limb-centric spherical space. A shoulder is located at the sphere center, with lines showing the connection to an elbow and hand. Blue and red dots represent the location where the sphere surface and projection of a ray from shoulder to hand coincide at a certain time step. Starting near the bottom, the series of dots shows the trajectory of limb motion for a person initially in a resting state through the beginning of a wave.

3.3.4 Software developments

During this period, we continued developing various aspects of our control software architecture for the Atlas robot. These have not all led to useable results, but we have learned a lot about the capabilities of the machine. In this section, we briefly describe various activities under this heading.

3.3.4.1 Higher-level joint PID loops

The standard BDI controller is hard-coded to use the input-side encoders for all joints. Unfortunately, the input-side encoders are not very accurate, and do not always provide the actual position of the joints. Therefore, we wrote our own joint position control code to use the output-side encoders, which are much more reliable. This new control code enables us to more accurately control joint positions and thus achieve more accurate Cartesian locations for the end effectors.

3.3.4.2 Characterization of feed-forward terms for torque and velocity

The combined velocity/torque output of each hydraulic actuator is a nonlinear function of the command signal, and differs for each joint. In addition, the distribution between velocity and torque varies depending on environmental factors. (For example, if you're pushing into a wall or a joint limit, velocity is forced to zero and everything is converted to torque.)

We collected data for each joint and produced functional models that take desired velocity and torque and produce an appropriate command signal. These models enable much better control of robot joint values. During this work, we found a discrepancy in the standard models for Atlas, where the steady-state torque predictions did not match observations.

3.3.4.3 MATEC Control Architecture

The Multi-Appendage Torque and Environment Contact (MATEC) control suite contains ROS-based software designed to perform precise multi-contact behaviors such as balancing, locomotion, and whole body manipulation. By incorporating this software suite into our control architecture, we will be able to switch cleanly between different controllers. The open-source code may be found at https://bitbucket.org/Jraipxg/matec_control (develop branch).

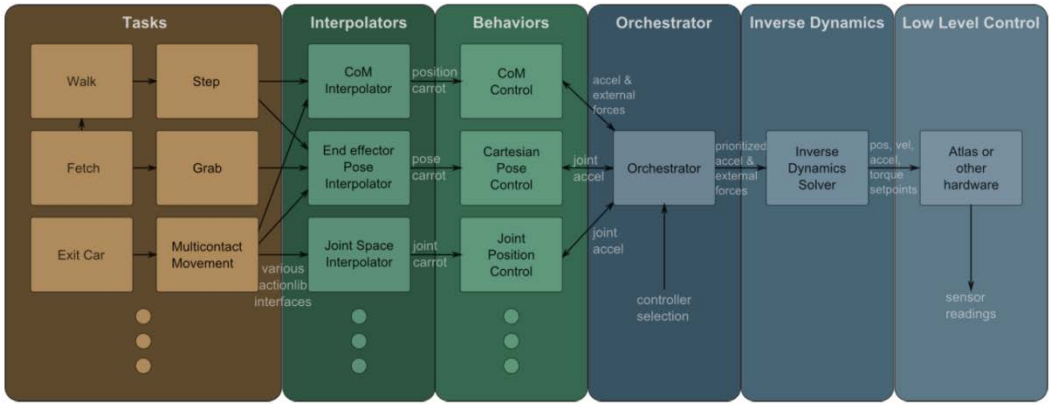

Figure 38: MATEC Architecture overview.

3.3.4.4 Overall architecture

MATEC control uses a distributed, layer-based architecture in which each component runs concurrently as a separate ROS node. This structure decreases computation-induced latency in layers closer to the hardware (e.g. the loop rate and latency of the inverse dynamics node is not affected by any higher level control that may take more time to compute) at the expense of increasing latency to layers farther away from the hardware. This is beneficial because the stability of fast low level controllers (e.g. dynamics compensation) is often more sensitive to latency than that of high level controllers / planners (e.g. planning what footsteps to take). Using this architecture also allows for more flexibility in terms of what control is running; New behavior nodes can be started / stopped on the fly without disrupting control so long as they notify the Orchestrator, and individual components can easily be exchanged as new algorithms are

developed. All communication within the architecture other than *actionlib* is done using shared memory for low latency. See Figure 38.

The current layer organization is as follows:

1. Tasks ("walk here" or "grab that thing there")
2. Open-loop Interpolators (Ensure that set points don't change faster than the hardware can handle)
3. Decoupled Behaviors (Joint position control, CoM control, etc.)
4. Orchestrator (Prioritizer that also coordinates with the behaviors to ensure smooth transitions between controllers)
5. Inverse Dynamics Controller (Ideally, makes the underlying system look like a bunch of decoupled double integrators to everything above)
6. Low level, robot specific control (Deal with unmodelled dynamics)

3.3.4.5 Tasks

Tasks are high-level state machines designed to accomplish a specific tasks, and are intended to be the primary interface to MATEC. Each task exposes an *actionlib* server interface which can connect to another task or external code. Every task connects to one or more interpolators and is responsible for notifying the orchestrator when it needs to switch between controllers.

A simple state machine for a "step here" task might include a pattern of four actions: **shift_weight → lift_foot → move_foot → place_foot**. Low level behaviors, like shifting the center of mass, are accessed via interpolators, each of which reports its status and completion state to facilitate correctly timed state transitions. Tasks can combined hierarchically as components in other tasks to create more complex, higher-level tasks. For example, "fetch me the coffee over there" is just a walk task, a grab task, and another walk.

3.3.4.6 Interpolators

The interpolator nodes expose an *actionlib* interface that allows the user to specify an array of time-stamped via-points that the underlying behavior should attempt to achieve. Each interpolator is paired with a behavior; though any given interpolator type can be used with many different behaviors (e.g. two instances of the same Cartesian interpolator node could be used to set the position of the CoM and the pose of an end effector). While a goal is active, the interpolator connects to its associated behavior via a streaming interface and outputs a mostly-open-loop "carrot" dictated by some interpolation method. A simple example of this would be a joint space interpolator controlling a single joint. An example *actionlib* goal might specify that the joint should move from wherever it is to angle 0.2 rad at time 1, then 0.5 rad at time 3 and finally 0.1 rad at time 6. The interpolator would stream the carrot to the behavior controller at a constant rate for a total of six seconds. The carrot will be equal to the goal value at the times specified in the *actionlib* goal, but will typically be different in between those via-points.

There are currently two interpolator types: Joint-space and Cartesian-space. As their names suggest, the joint-space interpolator interpolates a set of joints in joint space. The goal consists of a set of joint-space via-points for some subset of the joints assigned to the

interpolator. The Cartesian-space interpolator interpolates a pose in Cartesian space. The goal is simply a set of poses through which you want a particular ROS “tf frame” to pass. Interpolators can be constrained with maximum values for velocity, acceleration, and / or jerk, depending on the interpolation method. If these constraints are set below the actual physical limitations (e.g. even at max torque, a particular joint will not accelerate faster than 1.0 rad/s/s), it will be much more likely that the hardware will actually be able to achieve the goal. Currently only the maximum velocity constraint has been implemented for all interpolators.

This is also the category into which realtime teleoperation interfaces would fall. A teleoperation node should replace an interpolator, and send a carrot directly to a behavior.

3.3.4.7 Interpolation Methods

There are several interpolation methods that can be used by an interpolator. At present, the interpolation method is hard-coded, but there will eventually be a parameter or service interface. Trapezoidal velocity and doubles trajectories will be added in the future as well.

Minimum Jerk

This method uses a trajectory for jerk that is smooth except at the beginning and end and small in magnitude, which makes the acceleration, velocity, and position functions very smooth. This one is the easiest for a real actuator to follow, so it is currently in use in both nodes. Currently, this method is only set up to move to a single target and will ignore intermediate via-points in a goal array.

Linear

This method uses simple linear interpolation between the via-points and constant velocity between segments.

3.3.4.8 Behaviors

Behaviors are feedback loops that cause some desired behavior to occur. Each behavior controls a subset of the robot's joints and assumes that each joint's acceleration has been decoupled from the others (*i.e.*, no joint's acceleration will affect any others).

Each behavior must register with the orchestrator, provide a tare service, and provide joint accelerations for every joint it controls. Aside from those requirements, a behavior can implement any kind of controller. Currently, we have implemented tolerant Cartesian pose control and joint position control.

3.3.4.9 Orchestrator

The orchestrator handles controller prioritization and smooth controller transitioning. It attaches to all the behaviors and aggregates all their commands into a single whole-body dynamics command, containing position, velocity, and acceleration set points for every joint.

Priorities are positive integers, with smaller numbers indicating higher priority than larger numbers (zero is the highest priority). The combined command is currently created by copying the behavior command with the highest priority for each joint. Eventually a

more intelligent prioritization scheme might be used that smoothly combines behaviors with different priorities, possibly similarly to Luis Sentis's WBC.

Behaviors can be reprioritized via a service call to switch between controllers. For instance, to switch from joint position control to end-effector pose control on the same set of joints, you simply change the priorities such that the pose control behavior is higher priority than the joint position control. The

Orchestrator determines which joints will need to change controllers, and automatically tares the relevant behaviors to ensure a smooth transition.

3.3.4.10 Inverse Dynamics

The inverse dynamics solver is the core component of MATEC. It converts the decoupled accelerations and external forces calculated by higher level control into coupled actuator torques. This means that in the ideal case, it perfectly calculates the torques required to remove the effect of one joint's acceleration on another and removes the effects of gravity and external forces as well, essentially decoupling the joints. It will also try to calculate velocity and position set points from acceleration (currently by Euler integration) if they are not provided by higher-level control, since they are often needed for low level feedback control.

The current default implementation uses a Recursive Newton Euler Algorithm (RNEA) based inverse dynamics solver. Using RNEA is much more computationally efficient than solving the closed-form dynamics because it solves the problem recursively to avoid multiplying large matrices. However, such approaches are inherently model-dependent. If the model is incorrect, the torques output by the solver will not cause the desired joint accelerations. This results in more coupling between the joints than desired, potentially even more coupling than would be present if you had not attempted to compensate for the dynamics.

This can be overcome in many cases by integrators and / or high gain control, but the system as a whole will be much more stable the closer the model matches reality.

3.3.4.11 Adaptive Control

In order to improve robustness against modelling errors, MATEC contains a node that implements an adaptive control version of RNEA (still in development), based on Slotine's 1991 work in this area. There are twelve parameters adapted per link: six inertia values, three center of mass, one total mass, and two for coulomb and viscous friction. The adaptive law is based on both joint tracking error and torque prediction error and assumes that the robot's kinematics are known.

The node currently works very well for position regulation in simulation (i.e. going to a fixed position and staying there), exhibiting stable joint error convergence even when the adapted parameters are initialized within an order of magnitude of their true values (e.g. if the true mass was 10kg, the error will converge if the initial estimate is anywhere within [1kg,100kg]). However, the parameters rarely converge to their true values, even in simulation. Instead they converge to parameters that are statically equivalent at the set point. This often results in very poor behavior when trying to track a trajectory. An early form of the controller was shown to perform reasonably well on the real Atlas robot. The

joint positions got close to their set points (~0.005 rad error in the best case) and used much less feedback torque than a high gain PID loop.

Another downside of this adaptive controller is that it requires full state set points (position, velocity, and acceleration) for every joint. Traditional inverse dynamics solvers require only desired joint accelerations, so this places a greater burden on higher level control.

3.3.4.12 Low Level Control

Low level control refers to the interface to the actual hardware, which often involves some form of control loop to track a position, velocity, acceleration, or torque command. It is technically not a part of MATEC, but can affect its stability.

MATEC outputs a full state command containing positions, velocities, accelerations, and torques for every joint in the robot. The low level controllers should follow one or more state components, but don't necessarily have to track them all. For instance, MATEC is currently being used on Atlas by tracking the position and velocity set points instead of the torque set points.

The performance of all higher level controllers can be affected by the performance of these low level controllers. For example, given relatively slow torque tracking and nonzero latency, the torque commands sent from a PID controller may end up out of phase with the current torque produced, leading to oscillations and even instability.

3.3.4.13 Tolerant Cartesian space control

This algorithm uses a combination of inverse-Jacobean-based velocity solutions, as well as heuristics and a simplified random gradient descent. It is similar to our previous work, which used aspects of simulated annealing and gradient descent (SA-GD) to calculate real-time position and velocity commands that servo the current pose of the end effector to a desired target.

The target poses are specified in the global frame, so that movements caused by other active controllers (such as balancing), or external forces, are less-likely to affect the Cartesian task.

As with our previous SA-GD work, tolerance deadbands are introduced to reduce oscillations and instability near kinematic singularities.

3.3.4.14 Analyzed KDL

One of the recurring problems we have dealt with in various ways is the inverse kinematics for Atlas's arms. ROS is associated with an open-source kinematics library called the Kinematics and Dynamics Library (KDL: http://www.orocis.org/kdl). We have tested this extensively on Atlas. KDL uses the pseudo inverse of the Jacobian to solve for IK. We know that the pseudo inverse method is a least squares solution for computing the joint angles given the Cartesian coordinates and the partial derivative of each coordinate with respect to each joint angle variable. Since this is an approximate solution, KDL iterates until a Cartesian coordinate within permissible tolerance bounds is found. At each iteration, the Cartesian coordinate solution from the previous iteration is taken as the seed. KDL refers to this iterative method of determining the permissible closest Cartesian coordinate as the Newton-Raphson (NR) method.

3.3.4.15 Affordance Templates

After shipping the robot to BDI, we began updating our Operator Interface and exploring new user interface modalities. We successfully applied our Affordance Template (AT) approach to a simplified Atlas simulation, and are extending the AT framework to allow for multiple, customizable trajectories for each template. For example, a valve template can have a two-handed clockwise turn trajectory for a robot that can manipulate the object in front of it (i.e., R2), or a right-handed trajectory off to its side for a robot such as Atlas.

3.3.5 More software improvements

As we got closer to the Finals, and the hardware took on its final configuration, we made many incremental improvements to the software.

3.3.5.1 Low-level robot modeling and control

- **Improved dynamics simulation**
  - Modified gazebo to use a plugin and replaced the dynamics with a fast implementation of Featherstone's rigid body dynamics algorithms
  - Commanded torques are converted to accelerations and integrated using RK45 for significantly improved stability
  - The plugin also allows direct control of joint position and velocity (perfect control without dynamics) for easier testing of algorithms that only act on kinematics
- **Recursive dynamics compensation and kinematic evaluation on reconfigurable trees**
  - Graph-based representation of robot model allows fast generation of the tree models used by most dynamics algorithms
  - Loads from **urdf** model
  - Generated trees can be used to solve floating base hybrid dynamics problems (i.e. forward dynamics on some joints and inverse on others) and calculate kinematic / spatial motion / wrench transforms, center of mass, Jacobians, etc.
- **Sliding mode joint state observer**
  - Generates time-synced joint position, velocity, and acceleration trajectories intended to facilitate learning the dynamics model
  - Uses full robot acceleration prediction based on measured torques, with sliding mode terms for additional robustness
  - Successfully tracks ground-truth position, velocity, and acceleration trajectories in simulation, even with random noise added and induced mass/inertia modeling error up to 50% of the true values
  - Did not work as well in real life because both the dynamics model and torque sensing are inaccurate. Had to turn the sliding terms up so high to compensate that there was little actual benefit to using it over a simple low pass filter, other than acceleration generation

- **Reconfigurable robot modeling**
  - Set up model files so that we can change-out hands and other attachments by setting flags in a launch file
  - Enforced a consistent naming scheme for end effectors so that our manipulation code can adapt seamlessly when an arbitrary hand is used
  - The launch files also automatically spawn the appropriate drivers / sim plugins for whatever hardware is attached according to the model
- **Minimal-latency atlas hardware interface for improved low level control**
  - Rewrote our hardware interface to use fewer threads and streamlined code
  - Primary control thread now has a dedicated cpu core and is run as the highest priority thread on that system
  - Latency to the robot is around 3ms, which is close to optimal for the system

- **System identification for the newly upgraded Atlas.**

3.3.5.2 Operator Interface

- **Improved goal-space manipulation planning, partial integration with affordance templates** (see Figure 39)
  - Developed new planner node that incorporates both Cartesian interpolation and planning
  - Allows user intervention and visualization of plans
  - Supports a wider range of manipulation tasks than the previous incarnation and more fully supports goal-space planning
    - Commands consist of a set of segments
    - Each segment specifies a set of joints that the planner can use to move each of an arbitrary number of tools from an initial set of goal regions to a final set of goal regions in a synchronized, Cartesian-linear fashion
    - Goal regions are currently specified as 12-dof regions (min and max xyzRPY) around an arbitrary frame, but may later be extended to arbitrary polygons in 6D space
  - Interpolation is performed using minimum-jerk splining for extremely smooth Cartesian position, velocity, and acceleration trajectories
  - Plan is solved using a combination of Jacobian-transpose and random sampling techniques
  - Plan waypoints are generated at control rate (1khz) to ensure proper execution
  - Images show some planning paths (the teal boxes are the part it was able to solve, orange is the part that failed, the boxes visualize some of the goal-space regions that make up the plan) and viable dual-arm configurations (solved independently for now)

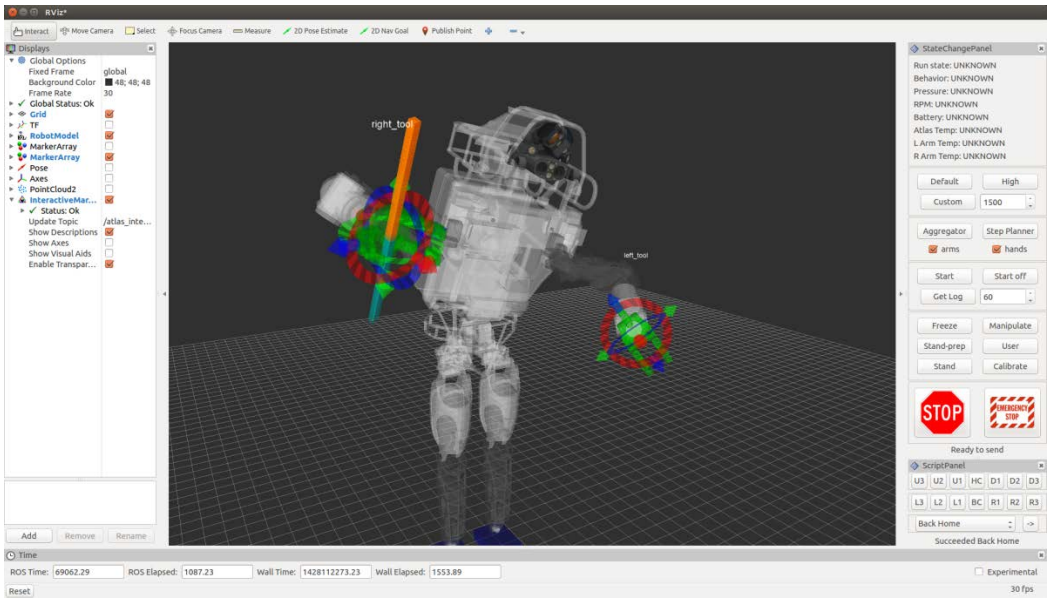

Figure 39: Goal-space planner interface using RViz.

- **Added RViz panels with buttons for simple commands**
  - The two panels on the right in Figure 1 (StateChangePanel and ScriptPanel).

3.3.5.3 Further description of important Operator Interface improvements in RViz

Interactive Controls GUI: Primarily worked on simplification of operator interface. This has included the consolidation of all RViz interactive markers for arm, hand, walking, and script-execution controls as well as the creation of a new, lightweight Interactive Controls GUI that can be added to the side of the RViz window. That gives a streamlined way of controlling the RViz interactive markers over using nested right-click menus. The Interactive Controls GUI provides dynamic of the following features:

- creating Cartesian 6-DOF RViz interactive markers for controlling the position of arbitrary chains,
- an interface for planning first, inspecting the resulting path via virtual robot overlays in RViz, and then executing if acceptable to the operator,
- the ability to mask certain joints out in the plan requests (for example, not using the back pitch joint when moving a hand to a location),
- the ability to set position and orientation tolerances from a set of acceptable bounds (e.g., ignore roll or orientation entirely, move within two centimeter of the specified goal, etc.),
- the ability to control how much feedback is displayed to the user when planning. For example the operator has the ability to request a visualization of just the final robot configuration or a visualization of the entire path it would take to get there. This will allow flexible response to bandwidth limitations.

Path Planner Abstraction: An improvement to the underlying interface of the IC GUI allows the operator to choose from multiple path planning algorithms/libraries to use for the robot. Current planners include our custom tolerance-based planner as well as MoveIt! This allows the front-end GUI and RViz markers to be robot/planner generic and should be able to be used on multiple platforms beyond Atlas. For instance, this approach has been tested on the Robonaut 2 platform using MoveIt! as part of other ongoing projects TRACLabs has with NASA-JSC.

Navigation/Footstep Path Planner Operator Interface: The Interactive Controls GUI also provides the ability to quickly drop navigation waypoints in the RViz environment, and plan paths (including footstep locations) through these waypoints. If the operator finds the path acceptable, they can submit it for execution. If not, waypoints can be adjusted, added or deleted. Individual footstep adjustments can be made based on sensor feedback in case the autonomous footstep planner makes mistakes.

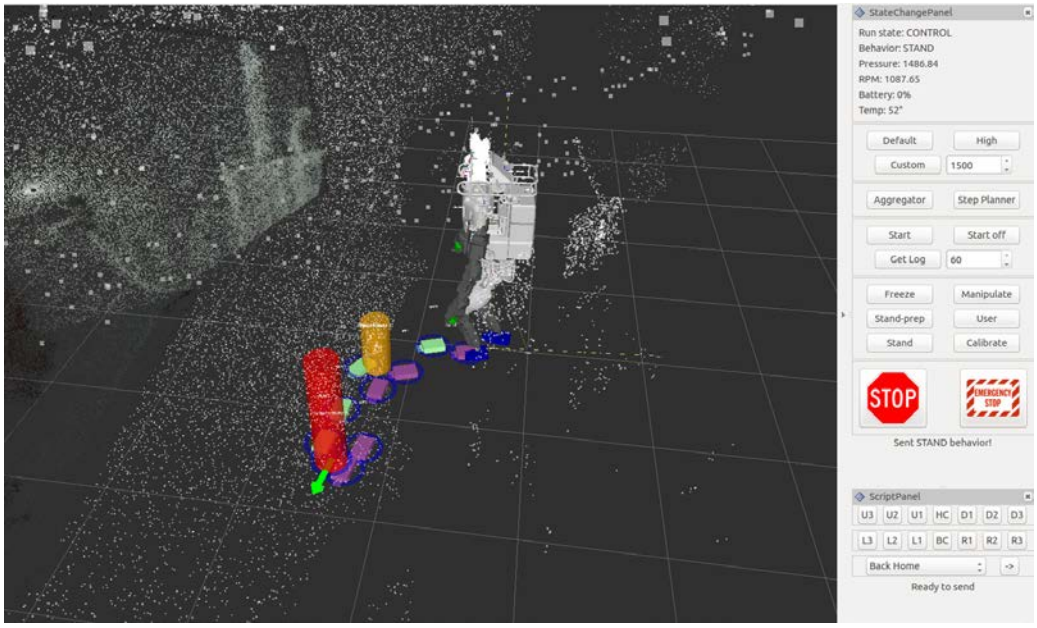

Figure 40: Footstep planning interface. The Operator places the target cylinders, and the planner produces a set of footstep placements. The Operator may adjust the footsteps as desired.

Affordance Template Improvements: The affordance template framework is being improved to handle the tolerance and joint mask features specified above, and is almost complete. Improvements have also been made to allow more flexible (and dynamic) hand allocation (in case hands break or are swapped).

3.3.5.4 Infrastructure

- **Improved handling of network issues and preparation for Finals**
  - Set up various elements of our code base to work under the prescribed network structure.

- Developed the transport for passing our data between computers, to support developing, deploying, and running the code across multiple computers.
- Set up internal lab network to the 10.6.2.x, 10.6.3.x subnets that can talk to each other the same way as at the Testbed in South Carolina.
- Set up *run scripts* that easily configure and startup the network shaping from a third computer on the robot-side of the setup.
- Set up a new low-level networking scheme to handle the low-bandwidth network.

3.3.6 Final software modifications

3.3.6.1 Executor

We developed an Executor capable of sequencing robot actions. These actions could be completely hard-coded, or could take parameters. The Executor enabled us to accomplish about half the tasks autonomously, after initial setup, but the Operator could intervene between steps if desired to tweak parameter values (a form of *adjustable autonomy*).

The Operator interface to the Executor had hotkeys that provided shortcuts for task marker placement, plus task-based constraints on Operator movement of the markers to reduce user error.

3.3.6.2 Manipulation planning

We developed a Goal-Space planner that allowed under-constrained task specification for more robust trajectory planning from various initial conditions. Manipulation stance locations were discovered by rapid random guesses until planning was completed successfully. Several pre-computed poses or pose transitions were stored as scripts and made accessible to the trajectory planner for speed.

3.3.6.3 Locomotion planning

Our flat-ground step planner optimizes the number of steps across three planning methods (turn-go-turn, Hermite spline, sidestep) and various parameterizations (stride width, turn out, starting foot) to find the fastest path to a specified goal.

For non-flat terrain, the user specified the angles for the foot placements. This was used in Egress and Rough Terrain.

3.3.6.4 Control

Our low-level control could be described as “Hydraulic-fluid-flow-based semi-recursive adaptively-gravity-compensated sliding mode joint space control augmented with PID, feedforward velocity, feedforward acceleration, and backlash-filtered embedded control.” However, not all control components were used with all joints.

We used semi-compliant joint position control, with mostly minimum-jerk-interpolated trajectories in Cartesian and joint space with acceleration and velocity bounds.

We calibrated joints manually at startup using the output-side encoders, and mostly operated at high pressure (2500 psi).

3.3.6.5 Visualization / OI

We used four monitors to display useful information to a single robot Operator. We displayed the SA cameras in upper monitors, but the primary tool for our OI is Rviz – the ROS visualization tool. We used multiple interactive panels for the various types of commands that we could send, and presented augmented reality in the primary Rviz window. Most data was visualized in 3D as point clouds or 3D overlays on images.

We used mouse-movable 6-DOF markers for manual tele-operation of end effector target positions, as well as footstep placement adjustments, and generation of multi-waypoint stepping paths. Keyboard shortcuts snapped the markers to the point cloud for faster Operator input.

A heads-up display (HUD) showed critical data, such as battery life, planner state, etc.

The "Prophesy" animation showed the results of combined locomotion and manipulation before plans were committed to action. A paper about Prophesy has been accepted at the 2015 Humanoids conference.

3.3.6.6 Networking

To manage the degraded network, we used two ROS masters: one on each side of the "long haul" link, and carefully controlled all data passing between them.

We used UDP on the burst link (mostly for perception data), and TCP on the small pipe (mostly for commands).

By streaming random samples of point clouds and re-aggregating the data on the OI side, we were able to provide decent data during blackouts.

3.3.7 Hardware Modifications

3.3.7.1 Polaris Modifications

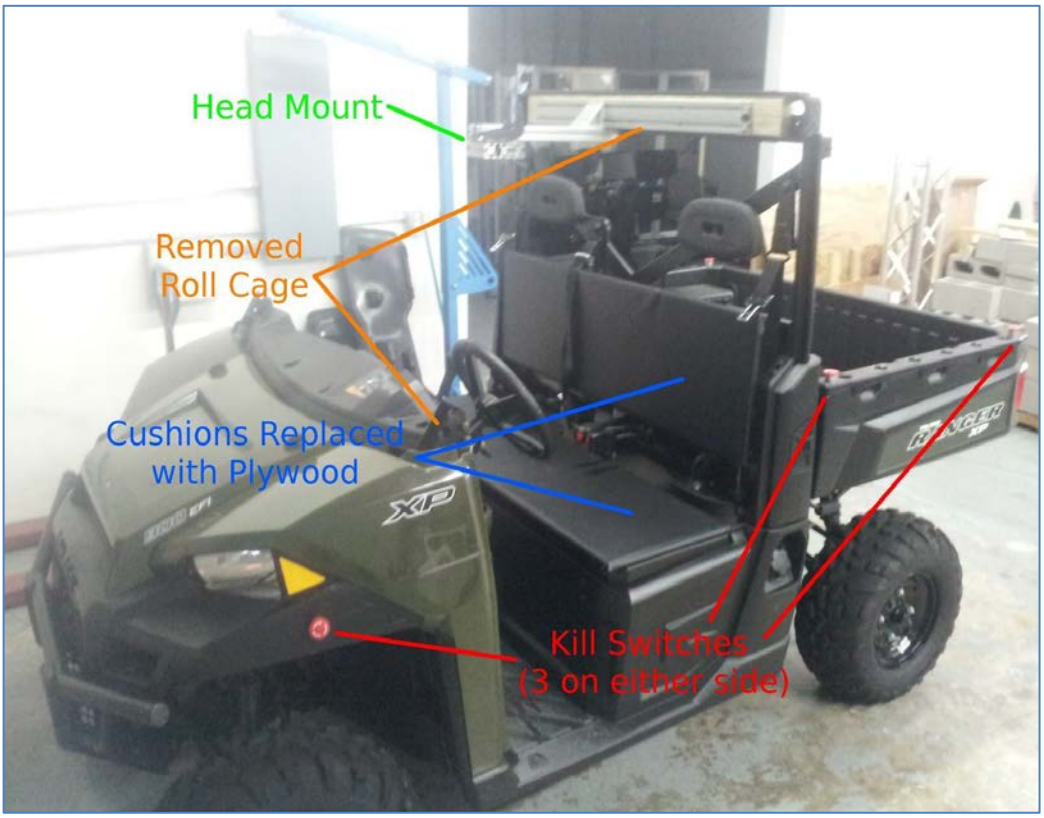

Figure 41: Polaris XP 900 with modifications for development of driving task algorithms.

We purchased a Polaris Ranger XP 900 EPS vehicle for development and practice for the Vehicle Task. This vehicle was modified to limit the speed of the vehicle to 10 mph. A wireless emergency stop (E-stop) was designed and installed for remotely disconnecting the main power from the batteries. E-stop buttons were installed around the perimeter of the vehicle.

The roll cage was modified to easily allow the robot to exit the vehicle with minimal interference. The seats were modified to allow additional room for the robot, and to allow the robot to easily slide along the seat for egress. The cushions of the original seats take up too much room and are too soft for the robot to move without being torn. The modified seats are ¾ inch plywood with a vinyl material cover.

A mount was constructed in the vehicle to support the head of the Atlas robot to simulate its position when the robot is sitting on the seat. This enabled us to gather sensor data without needing the full robot in the vehicle.

3.3.7.2 End Effectors

We developed a compliant “Pogo” hand for the left arm (See Figure 42). This consisted of two nested pieces of PVC tubing, approximately 9 inches long, with a bungee connecting them. This hand was used for opening the door, pushing the door (if necessary), turning the valve, turning on the drill, and flipping the breaker switch. We

investigated a passive left hand fairly early on, because the power pass-through cable in our left forearm did not work. After a time, we felt we had a satisfactory passive solution, so fixing this aspect of our left forearm became low-priority for BDI.

We used a Robotiq hand on the right arm. This was used for steering and holding the drill in the wall task. We felt it was too unwieldy for the door handle or small valve handles.

3.3.7.3 Car Modifications

The driving and egress tasks were likely to be the most difficult tasks for our team, as we do not have the expertise or resources to develop the software necessary to perform these two tasks without major hardware modifications. These two tasks are especially difficult because our robot was never designed with the ability to sit down, let alone get up from sitting, in mind. The vehicle chosen by DARPA is relatively small, with less legroom than a typical sedan, adding to the challenge. We aimed to solve most of the technical challenges associated with these tasks with hardware, rather than software.

We chose to forsake driving by any obvious, conventional, or visually appealing means in favor of making the egress task as easy as possible, as the egress would be the most difficult out of any of the tasks. Based on the specifications provided by DARPA, we chose to place the robot sitting sideways with legs outside of the vehicle and feet on an added step, so that the robot can simply transfer its weight onto its feet, stand up, and step out. To help keep the robot in place during driving, we added a "booster seat" in the form of the robot's "buttocks" (or rather, the robot's "lack of buttocks").

After establishing how and where the robot will be situated to best accomplish the egress task, the next step was to bring the driving controls (steering wheel and gas pedal) over to where the robot can reach them. The steering wheel controls are placed near the robot's right shoulder. A 1:1 transmission made out of a combination of regular and flexible drive shafts, mounted by bearings on wood connects the steering wheel to a custom handle that the robot can grasp. The accelerator pedal controls are placed where the robot's left arm can easily rest. Another custom handle connects to the drive pedal through a bicycle brake cable.

Testing of the egress task showed us that it would be far more reliable if we added another step in addition to the first. The second step folds up to decrease the car's width while driving, but can be deployed by having the robot simply tear some masking tape that holds it in place. Upon being deployed, the second step falls to the ground.

3.3.7.4 Miscellaneous

We added a shoulder webcam to assist in driving by providing the Operator with a view of the steering wheel.

We used extension cables to move the wireless access point antennas away from critical areas, so they would not hit the door frame or get tangled in the belay rope.

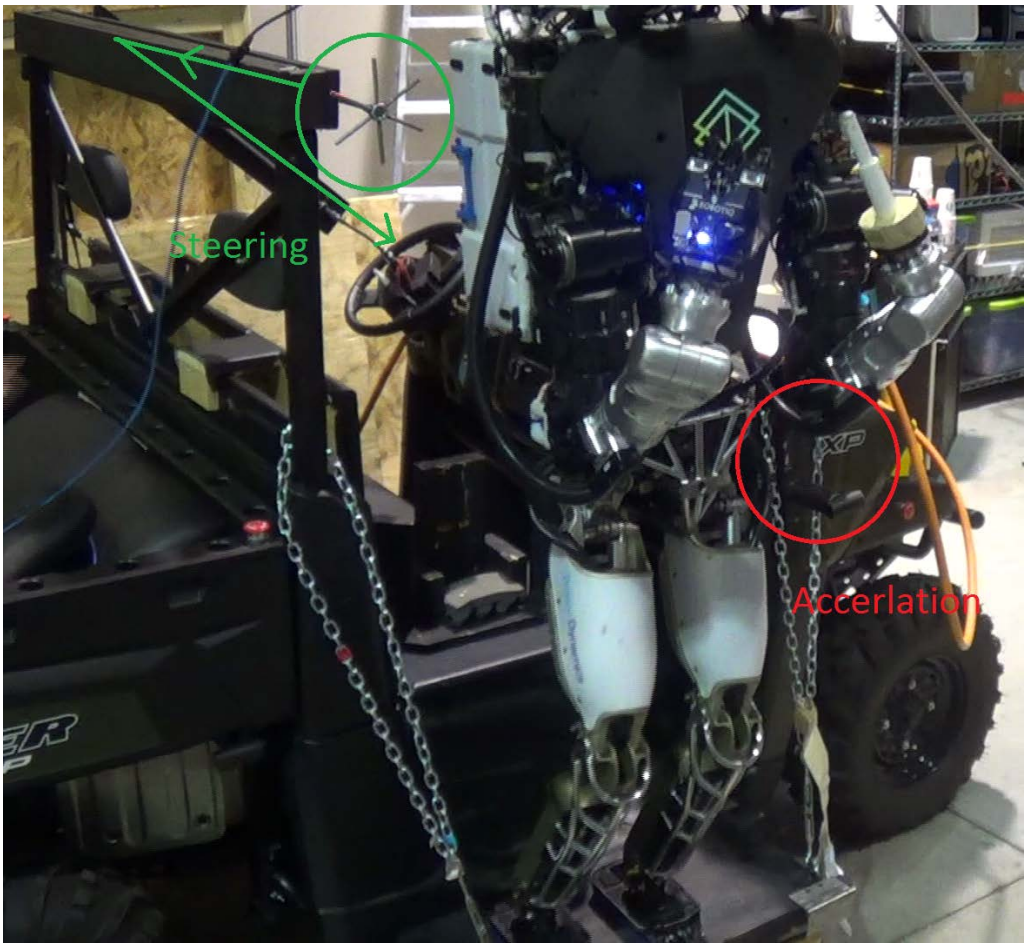

Figure 42: Atlas balancing on the upper step added to the Polaris. Marked on the left are the attachments for steering control. On the right are the attachments for accelerator control.

3.3.8 Grasp affordances and autonomous reach

Researchers from our partner, Northeastern University, implemented a new perception algorithm on the Atlas robot for localizing handle-like grasp affordances in three dimensional point clouds. The algorithm searches point clouds for neighborhoods that appear graspable, checking that there is sufficient clearance around them to accommodate an enveloping grasp. The advantage of this kind of algorithm is that it does not require foreknowledge of what the object it is picking up looks like. It is only concerned with finding locations that are amenable to being grasped. Figure 43 shows an example of these grasp affordances found on a DRC ladder.

With this new algorithm, we were able to create a semi-autonomous grasping solution that was integrated into the TRACLabs DRC OI to reduce the time it takes for an Atlas operator to pick something up. With the press of a button, the user can command the robot to search for grasp

affordances in the robot's workspace. The robot then returns the grasp orientations it finds, and the operator has only to select one and hit the send button to trigger an automated grasp action plan. The plan includes the choice of which arm to use, commands to open and close the hand at the correct times, and the proper trajectories to follow to avoid hitting the object with the hand's fingers on the approach. Figures 44 and 45 show the new UI section for automated grasping in action. Figure 46 shows the robot successfully acquiring a drill using grasp affordances.

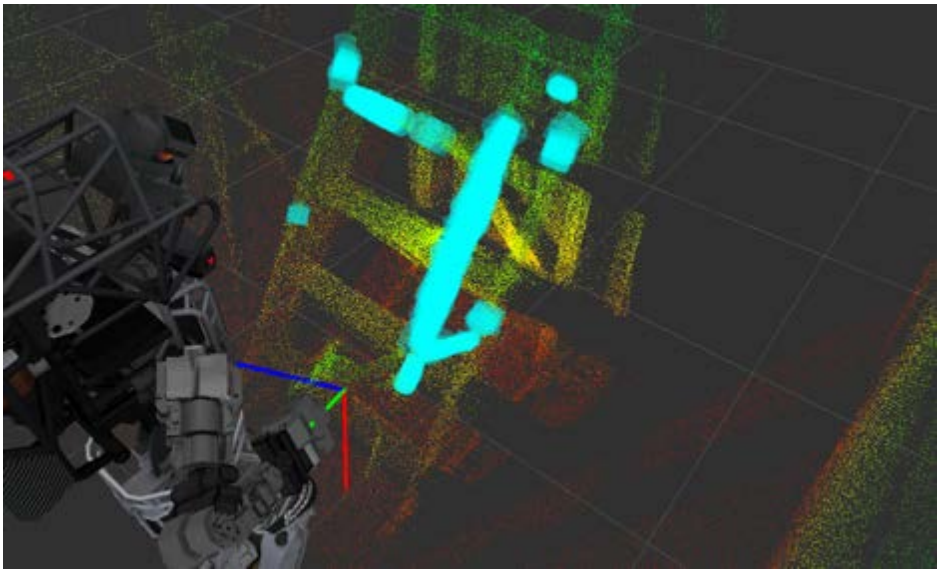

Figure 43: Grasp affordances found on a DRC ladder. Without the handle detection/grasping algorithm, the operator would have to manually position a six degree of freedom marker several times to fine tune a grasp configuration, and manually command the hands to open and close at the appropriate time.

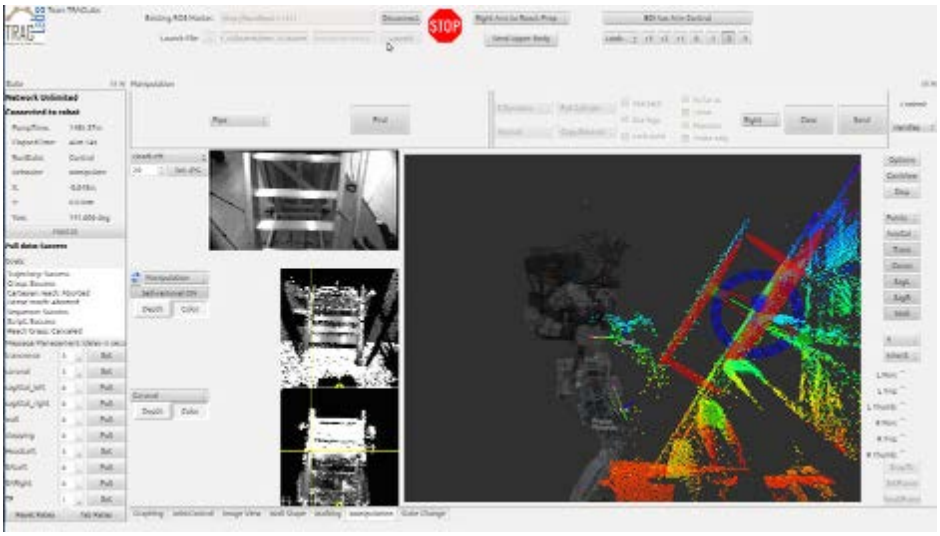

Figure 44: Grasp orientations for the operator to choose from on the DRC ladder.

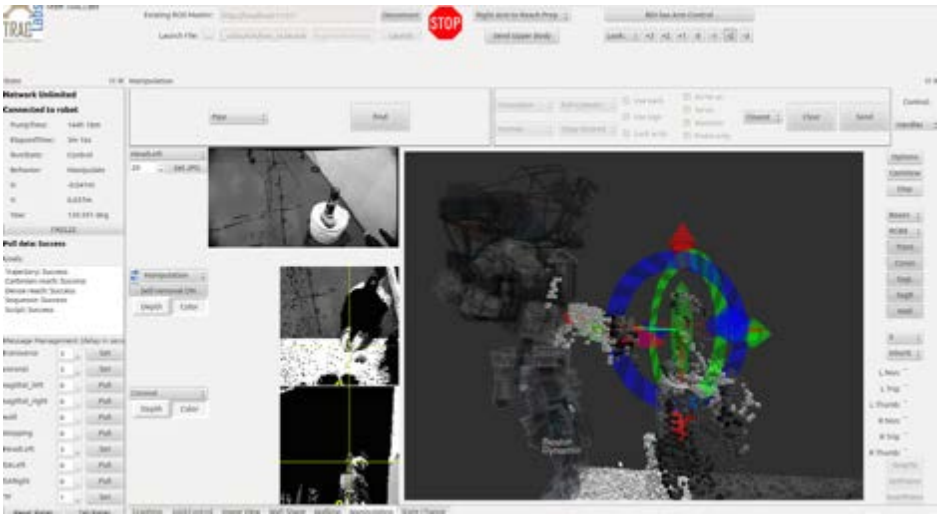

Figure 45: Grasp orientations on a DRC drill.

Figure 46: Successful automated grasp of a drill.

3.3.9 Driving task

As mentioned above, we first situated the robot in a pose in the vehicle that we believed we could egress from, and then brought the vehicle controls to the robot via two rather "Rube Goldberg" mechanisms. It was very important that the robot could engage and disengage the steering and accelerator mechanisms by itself.

3.3.9.1 Steering

The steering mechanism used the rotation of the right arm wrist roll joint to affect the angle of the steering wheel via a transmission made of wrench extension rods. Because

there was significant windup in the system, the robot had to be able to disengage the mechanism, "unwind" the joint, and re-engage to turn it further. It was generally a good idea to move the vehicle a bit too, to allow the windup torque to settle through the steering wheel. We used a piece of tape on the steering wheel, along with a shoulder-mounted webcam to gauge how far the wheels were turned. The Operator had a tool that allowed him to type in a steering wheel angle and visualize the arc that the vehicle would follow superimposed on the sensor data in Rviz.

We found that the hand by itself sometimes became jammed with the tines of the steering mechanism, so created a "bear claw" block that could be held that had spokes that would engage the spokes of the steering mechanism. This block was discarded after completion of the driving task.

The Operator used keyboard-activated joint scripts for controlling steering and regrasping.

3.3.9.2 Accelerator

The accelerator mechanism used the left forearm placed inside a hook-shaped handle. When the robot moved the forearm down, it pulled the bicycle brake line, which depressed the accelerator pedal. We relied on the fact that the vehicle quickly comes to rest when the accelerator pedal is not being pressed.

The Operator used keyboard-activated joint scripts for different strength and duration bursts of acceleration. By observing the response to different scripts, he was able to determine which ones produced the right size "quantum" motions.

Our process of driving was very much "bump-and-wait": Choose a steering angle, choose an accelerator script, then wait to see the result… then repeat!

Because there were too many cars and people in our lab's parking lot during the day, we decided to test driving in our parking lot at night. For three nights prior to shipping the robot, we took the robot outside and worked on our ability to drive with the steering and accelerator mechanisms. Several modifications came out of this for both hardware and software. The bear claw and webcam were added at the Finals.

3.3.10 Egress task

The robot sat side-saddle in a "booster seat" that was designed to constrain the pelvis from sliding in any direction except up and off the seat. The feet rested on a step that attached in the place of a running board. During testing in our lab, we found that we could run joint scripts that caused the robot to "get up" and balance on its feet on the step. At that point, we could transition to BDI's balancing controller. From there, we could use BDI's stepping controller to step down. This all worked well up to having the first foot on the ground, however, as the robot lifted the back foot to step down with that one too, the vehicle's suspension lifted up as nearly 400 pounds weight was lifted off the step. This caused the step to continue applying an upward force on the back foot in a manner not expected by the stepping controller, and it was not able to handle it. We found however, that we could successfully step off the first step if we had a second platform on the ground – about the height of a cinder block.

Based on this discovery, we developed (at the Finals) a second platform that swung down and fell to the ground when released. We used masking tape to lightly secure the second platform up and out of the way of driving until the robot reached down at the end of the drive and broke the tape. When that happened, it swung down and fell to the ground. Cables prevented it from falling too far from the first.

After some experimentation, we found that the first footfall from upper to lower platform needed the robot's foot to be splayed outward at just such an angle so that the center of mass transitioned along the main axis of the foot as it stepped. If this was not the case, the robot tipped over. We practiced this many times using our Polaris in our garage, but the DARPA Polaris was a slightly different model.

3.3.11 Door task

Although this is an outdoor task, and thus has good comms, we were able to create a script that covered the major steps of this task. To begin, the Operator matches a "door task" template to the sensor data. When he triggers the script, the robot:

1. walks to a pre-set location a fixed offset from the marker that is good for manipulation,
2. moves the left "pogo" hand near the door handle,
3. swipes downward to open the door,
4. pushes the door open (if necessary),
5. walks to a second pre-set location,
6. tucks arms to create a narrower profile,
7. walks to a third pre-set location that guarantees good steps through the door,
8. walks to a fourth location, on the other side of the doorway.

Although the robot sways a little from side to side when using BDI's quasi-static "stepping" controller, the third pre-set location makes sure that right foot steps on the threshold, and thus the sway happens around the left-side door frame.

The Operator has the ability to jump in and adjust marker and foot placements throughout the script if desired.

3.3.12 Valve task

We were able to get a good sensor reading of the valve task while still outside the door and getting good comms. Thus, our Operator was able to start matching the template to the sensor data before the network shaping was turned on.

Similar to the Door task, the Operator uses a "valve task" template to quickly align to the sensor data. When the script is triggered, the robot:

1. walks to a pre-set location a fixed offset from the marker that is good for manipulation,
2. moves left "pogo" hand near the valve handle,
3. inserts pogo into a (user-specified) gap in the handle,
4. presses against the outer rim of the handle,

5. moves more than 360 degrees is a circular motion
6. extracts the pogo.

Although the network shaping causes multi-second blackouts, the stepping controller is slow enough that the Operator has opportunities to adjust markers if desired. Furthermore, since the robot is running autonomously, the Operator can start placing markers for the next task.

3.3.13 Wall task

Once again, we used a script based on markers for drill and wall placement. Once again, the Operator has the opportunity to edit markers on the fly if desired. This is one of very few task activities that require a grasp, so we used the Robotiq hand. The script:

1. walks to a pre-set location a fixed offset from the drill marker,
2. picks up the drill with the Robotiq hand on the right arm,
3. moves away from shelf, loosens and tightens grip to settle drill in grasp,
4. stabs with pogo in a varying pattern until Operator triggers next step,
5. extends left arm for balance, moves drill near wall,
6. cuts circle – more than 360 degrees,
7. uses drill to punch circle,
8. deposits drill,

We developed and tested this script while at the Finals, after seeing the relative locations of drill shelves and cutting wall. Notably, that there were probably places we could stand where we could pick up the drill, activate it, and then cut the wall without needing to reposition the feet. By using markers, the Operator can accommodate different heights of shelf or cut-out pattern, as well as orientations of the drill.

3.3.14 Mystery task

Once again, a script based on markers allows the Operator to set things in motion based on historical sensor data, regardless of the blackouts.

It should, perhaps, be emphasized here that these marker location and trigger commands are small, and thus are sent via the “small pipe”, which is not subject to blackout. So as long as there is not much sensor drift, the previous point cloud sensor snapshot is usually good enough for the Operator to use.

After the Operator has placed the “breaker task” marker, the script:

1. walks to a pre-set location a fixed offset from the marker that is good for manipulation,
2. moves left “pogo” hand near the breaker handle,
3. moves pogo along preset “swipe” motion – farther than necessary,
4. retracts arm.

3.3.15 Terrain/Debris task

We had no interest in the Debris task, because we only had one gripper (and one pogo), and shuffling through the debris would be too risky – we did not want the robot to fall. Our strategy for traversing Terrain, on the other hand, was very robust and solid. In practice, it worked just about every time. However, there were some variables left to the discretion of the Operator, and thus subject to possible Operator error.

The Terrain task was broken into three scripts: (1) ascent gets the robot onto the first set of bricks; (2) traverse gets the robot across the middle bricks; and (3) descent gets the robot off the final bricks and back to the ground. For each script, footstep placements are laid out, and the Operator can then adjust them if necessary.

3.3.16 Stair task

This is an outdoor task (and thus has good comms), but again we were able to speed up execution of the task by using a template and the sequencing Executor. The Atlas robot is not particularly flexible, and we have limited control over the details of how the BDI stepping controller operates, so there was a lot of experimentation to determine how to get the legs to manage a flight of stairs without the shins hitting the next step. The script:

1. tuck arms and lean forward slightly (biases pelvis back away from stairs),
2. walk to preset location relative to stairs,
3. step first with left foot at angle, with heel hanging off stair a bit,
4. step second with right foot nearly straight, heel must hang off,
5. repeat four times!

4 RESULTS AND DISCUSSION

4.1 VRC

4.1.1 Walking task

Using our crawling algorithms, we were able to complete all stages of the course in about 16 minutes with minimal upload. We scored all 20 possible points in this task: five runs, four points per run. Although we occasionally triggered the fall event, it never happened three times in one run, so it was not a problem.

Although successful, these scripted behaviors were clearly not going to work with the real Atlas hardware (or any hand hardware), so this was an example of engineering a solution to the problem of scoring points, not the problem of controlling a live Atlas.

The OI tabs that were developed for locomotion – especially the ones that enabled path planning for the BDI walking behavior and the one that allowed development of scripted behaviors – were useful for all the VRC tasks, and carried over well to the DRC Trials.

4.1.2 Driving task

Because the scripted "swan dive" behavior did not have a very large capture envelope of standing positions next to the vehicle, our Operator missed once, which caused the robot to fall. Because there was no lateral situational awareness for the robot, it because wedged under the vehicle in a manner that we could not easily extract from, and when we finally did, we were not able to stand up. This, in turn, turned out to be because the hands were not in the correct pose for the "standup" script. We triggered many falls attempting to stand during that run.

These failures pointed to two places where some automation could have helped:

1. The OI could have validated the starting pose relative to the vehicle, and not allowed the Operator to trigger the Ingress unless the pose was within the capture envelope.
2. The OI could have identified that the hands were not in the correct configuration, and either prevented the Operator from triggering the "standup" script, or caused it to set the hands correctly at the start of the script.

These sorts of intelligent assistance in an Operator Interface are part of the field of Adjustable Autonomy, and provide benefit when the system is capable of sensing enough to know how the current state of the system relates to the current task.

We scored the first point in the other four runs, for a total of four points in this task.

4.1.3 Hose task

As expected, we successfully picked up the hose quite quickly every time. However, our OI did not provide good enough control over the pose of the nozzle for our Operator to robustly mate the nozzle to the receptacle, and he was only able to accomplish the mating once.

This was a situation where we needed to use perception-based automation. We needed the system to analyze the point cloud and match a model to the receptacle, effectively giving us the location of its primary axis. There are many ways this could have been done – with more or less Operator involvement. With that information, and similar information for the nozzle, the system could have served the nozzle into the receptacle easily every time.

Five pickups and one mating gave us six points on this task.

4.1.4 Scoring results

The top-scoring team, IHMC, accumulated 52 points. They clearly had good success with every task. They did most of their development in their own simulation environment, and did not rely on BDI's walking or balancing behaviors.

Next, WPI (Track C) scored 39. Early on, they found a method of successfully entering the vehicle in a pose that allowed driving, and thus were able to accumulate a lot of points of the Driving task, which most of the rest of the field struggled with.

Next, CMU scored 34 points.

TRACLabs' 30 points ranked fourth, just edging out JPL, who scored 29 points, and TORC, who scored 27.

Five more teams scored in the 20s, one scored 16, and the remaining ten teams scored in the single digits.

Frankly, we did not expect getting only 30 of 60 available points to score very highly, and were quite surprised that this was good enough for 4th of 22 teams. We believe that the network throttling stymied a lot of teams, and that many teams struggled to negotiate the Walking task by walking upright, rather than discovering that it could be traversed using prone positions.

4.2 DRC Trials

4.2.1 Vehicle task

We did not attempt this task and therefore scored no points. Four of the 16 teams successfully scored one point for driving. None attempted egress.

Figure 47: Team TRACLabs performing the Terrain task at the DRC Trials. (a) Traversing the ramp; (b) traversing the flat cinderblock staircase.

4.2.2 Terrain task

In practice, we were able to accomplish all four of the subtasks: ramp, herringbone cinderblocks, flat cinderblock staircase, and slanted cinderblock staircase. Unfortunately, we fell early and had an intervention at 7:20. We scored one point in this task at 15:08. When we were at the top of the flat staircase and starting stepping down, the robot lost its footing and fell. We were not able to determine the cause, although there was a wind-tunnel effect between the two conex boxes that were supporting the belay rigging – the top of the flat staircase was right in the middle of the gap between those structures. See Figure 47.

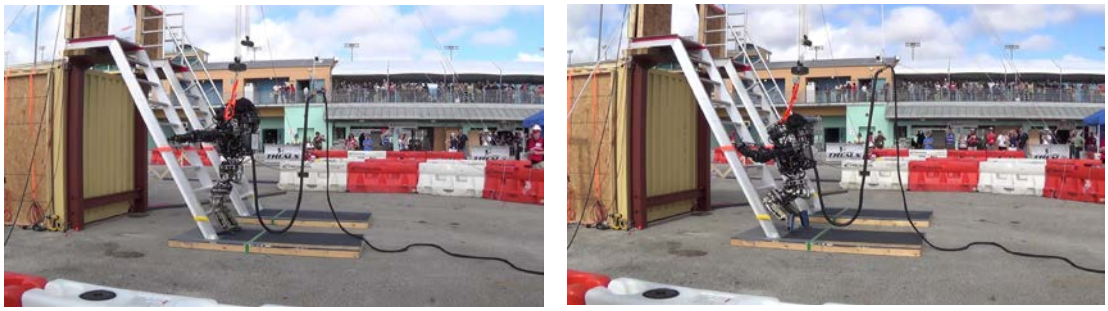

Figure 48: Team TRACLabs performing the Ladder task. (a) Setting the hook hands on the fourth step. (b) kneeling on the first step.

4.2.3 Ladder task

Our method for kneeling on the first step using the hook hands was very robust, and we had no trouble scoring the one point on this task. See Figure 48.

Figure 49: Team TRACLabs performing the Debris task at the DRC Trials. (a) Executing the first "plowing" attempt; (b) preparing for a second "plowing" attempt after an intervention.

4.2.4 Debris task

In the Trials, the first plowing pass successfully removed 5 pieces of wood, ending with a fall and an intervention, as expected. After an unsuccessful second plowing attempt and intervention, it was decided that the remaining debris could not be cleared.

We scored one point with an intervention at just over one minute. We had another intervention at 9 minutes and ended the task at that point. See Figure 49.

4.2.5 Door task

We successfully opened the push door and walked through it at 18:46. This was worth one point. We opened the second door, but the wind shut it before we could get in position to walk through it. See Figure 50.

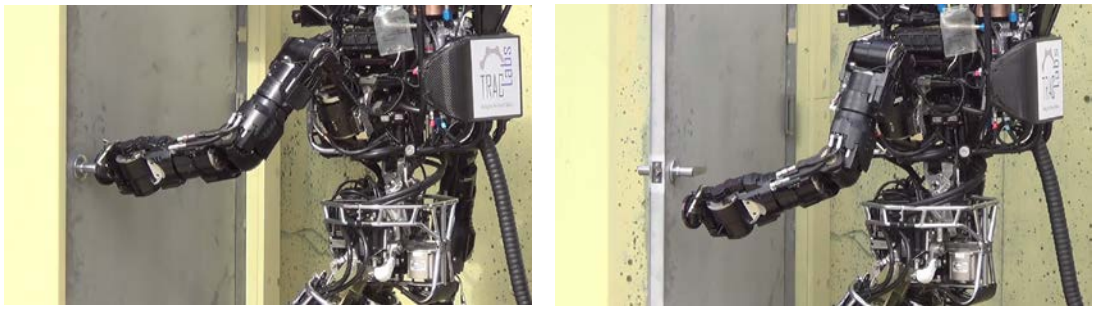

Figure 50: Team TRACLabs performing the Door task. (a) Unlatching the pull door; (b) the pull door is open.

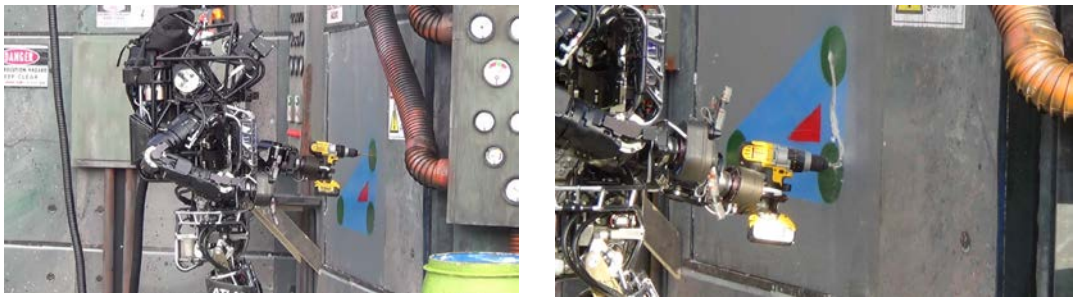

Figure 51: Team TRACLabs performing the Wall task. (a) Preparing to start the first cut; (b) finishing the first cut.

4.2.6 Wall task

We scored one point in this task in just over 23 minutes. We went out of bounds while attempting the second cut, and then ran out of time.

For this task, we needed a better way to keep the orientation of the drill perpendicular to the wall. The iRobot hand did not grip the handle tightly enough to prevent it from rotating, which caused the cut to drift to the side, as seen in Figure 38(b).

Figure 52: Team TRACLabs performing the Valve task. (a) Turning the lever valve with the right hand; (b) turning the small handwheel with the left hand.

4.2.7 Valve task

We scored all three points for turning valves in just over 27 minutes with no interventions, which earned the fourth (bonus) point. This was by far the easiest of the

eight tasks. Of the 13 teams that scored any points at all at the Trials, nine scored four on this task. See Figure 52.

Figure 53: Team TRACLabs performing the Hose task. (a) Grasping the hose; (b) walking the hose nozzle to the wye; (c) attempting to thread the nozzle.

4.2.8 Hose task

We scored two points in this task in approximately 18 minutes, spending the remaining time unsuccessfully attempting to attach the hose. Like the Valve task, most top teams scored multiple points on this task. See Figure 53.

4.2.9 Scoring

Team SCHAFT clearly mastered this competition, scoring 27 of the 32 available points. IHMC was next with 20, CMU had 18, MIT had 16, and JPL scored 14.

We scored at least one point on all seven tasks we attempted. Three of the fifteen other teams did this. We scored perfectly only on the Valve task, and got a second point only on the Hose and Valve. With eleven points, we ranked sixth of the sixteen teams that competed, beating WPI, who also scored 11, in the tiebreaker (number of interventions).

Team TROOPER rounded out the top eight with a score of nine.

Although we expected Track A teams to dominate this competition, and SCHAFT certainly did, five of the top eight teams were Atlas teams. This speaks well of BDI and the support they gave the Atlas teams, and also of the DARPA team, which kept the playing field fair between the two groups.

As with the VRC, we found it surprising that more teams did not score higher – were not able to score in the double digits. We felt that it was within our reach to score more points on Terrain, Door, and Wall. However, with only one official attempt at each task, we were happy to score as well as we did. It would not have been reasonable for us to expect to score more than three more points, so our rank was appropriate. In fact, the rankings were quite similar to the VRC rankings for Atlas teams, with the exception that we edged WPI this time, and JPL beat us (of course, this was their Track A team, not their Track B team).

4.3 DRC Finals

Figure 54: Team TRACLabs driving the vehicle toward the End Zone.

4.3.1 Vehicle task

During the Dress Rehearsal on 6/4, we attempted to drive three times and hit barriers three times, the last of which damaged our step.

For the First Finals Run on 6/5, we added the bear claw and the webcam, and successfully drove to the finish line in about 20 minutes.

During the Second Finals Run, we inadvertently opened the hand and dropped the bear claw while driving. This led to the steering mechanism getting jammed in the hand, and then a re-grasping action detached the extensions. We called for a reset, reset the apparatus, and then successfully drove to the finish line in about six minutes. See Figure 54.

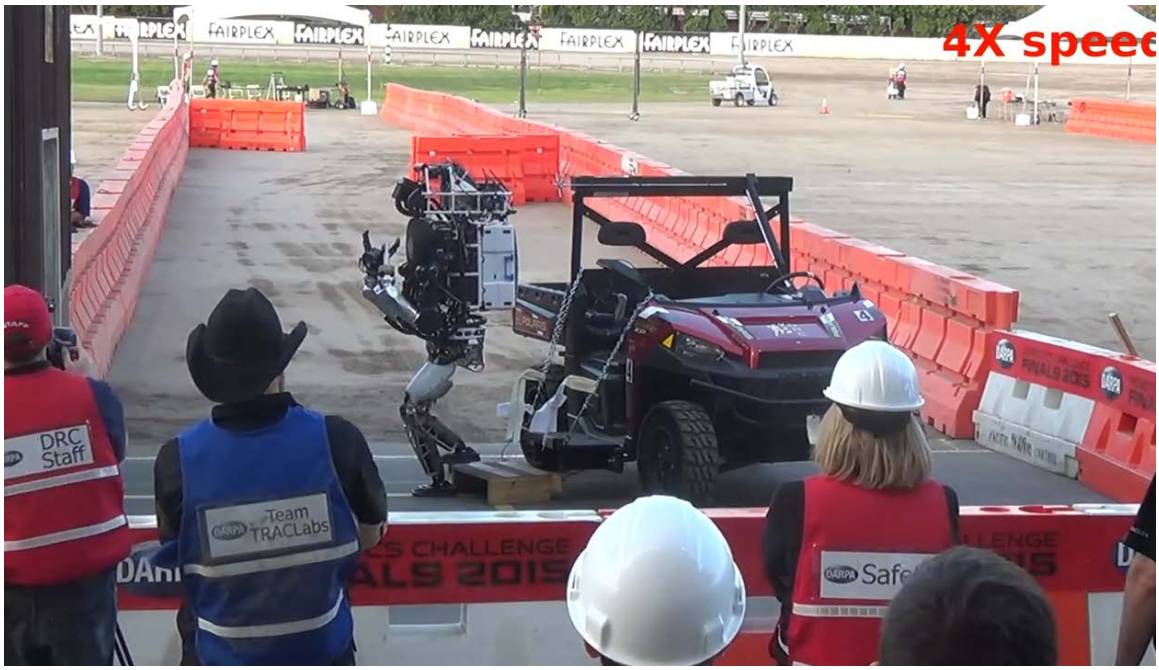

Figure 55: Team TRACLabs egressing the vehicle.

4.3.2 Egress task

During the first Finals run, we successfully accomplished the egress after driving. This was our second point, and came after roughly 25 minutes.

During the Second Finals run, the first footstep placement down was a little further than the robot control system could handle, which resulted in it planting the foot closer in than anticipated. When the second foot stepped down, the second heel overlapped the first by a fraction of an inch, which was enough for the control system to get thrown off and the robot to fall over off the step. This ruptured two hydraulic lines and ended the Second Finals Run. See Figure 55.

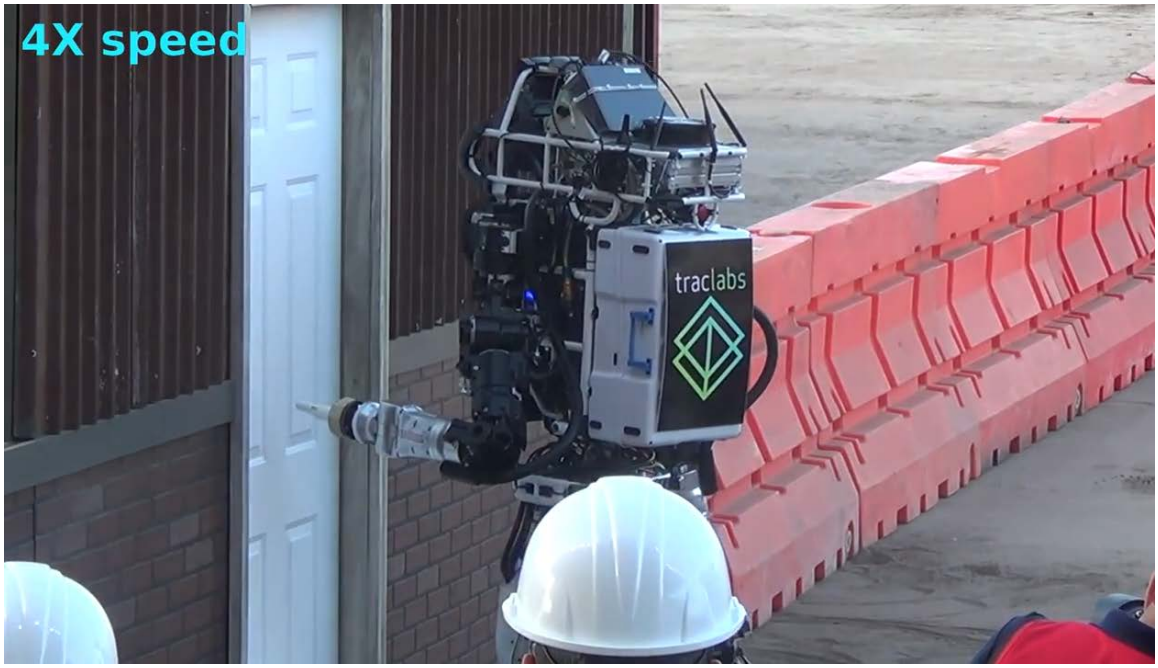

Figure 56: Team TRACLabs approaching the door.

4.3.3 Door task

On the first Finals run, we successfully opened the door and walked through it. This was our third point, and came after roughly 35 minutes. See Figure 56.

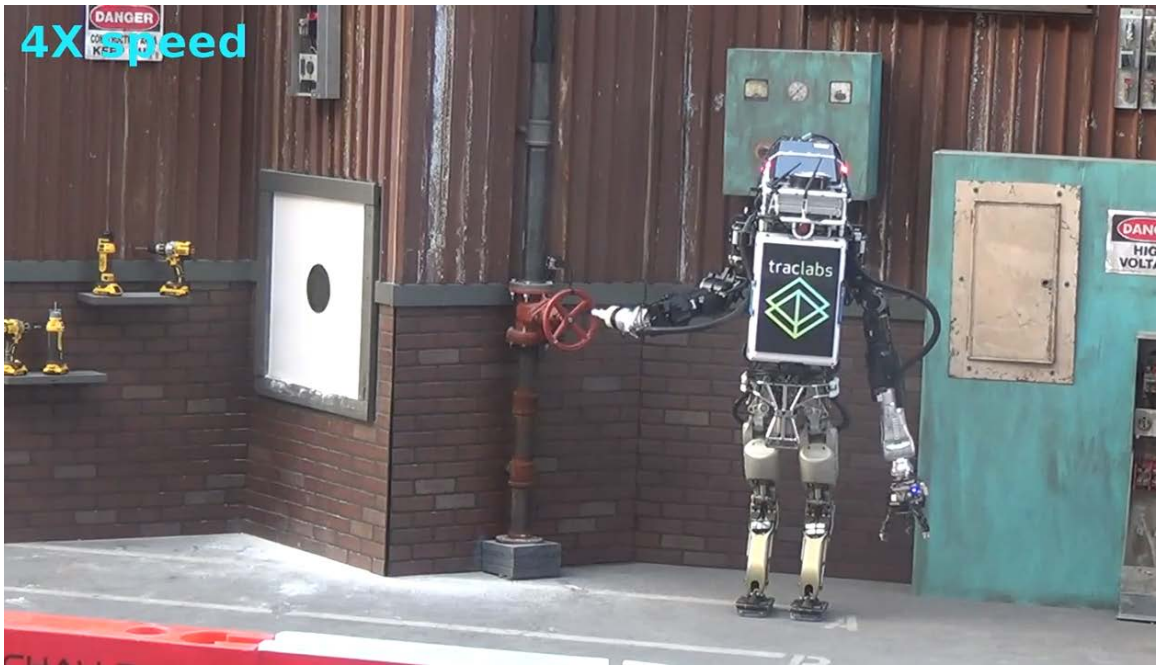

Figure 57: Team TRACLabs approaching the valve.

4.3.4 Valve task

During the first Finals run, we successfully turned the valve. Because we align markers to sensor data prior to committing, the network shaping really did not cause much trouble to the system. This was our fourth point, and occurred at roughly 43 minutes. See Figure 57.

4.3.5 Wall task

We bypassed the Wall task in the interest of time, although we had had some decent success with this task in practice. We were planning to attempt it during the second Finals run, but never got there.

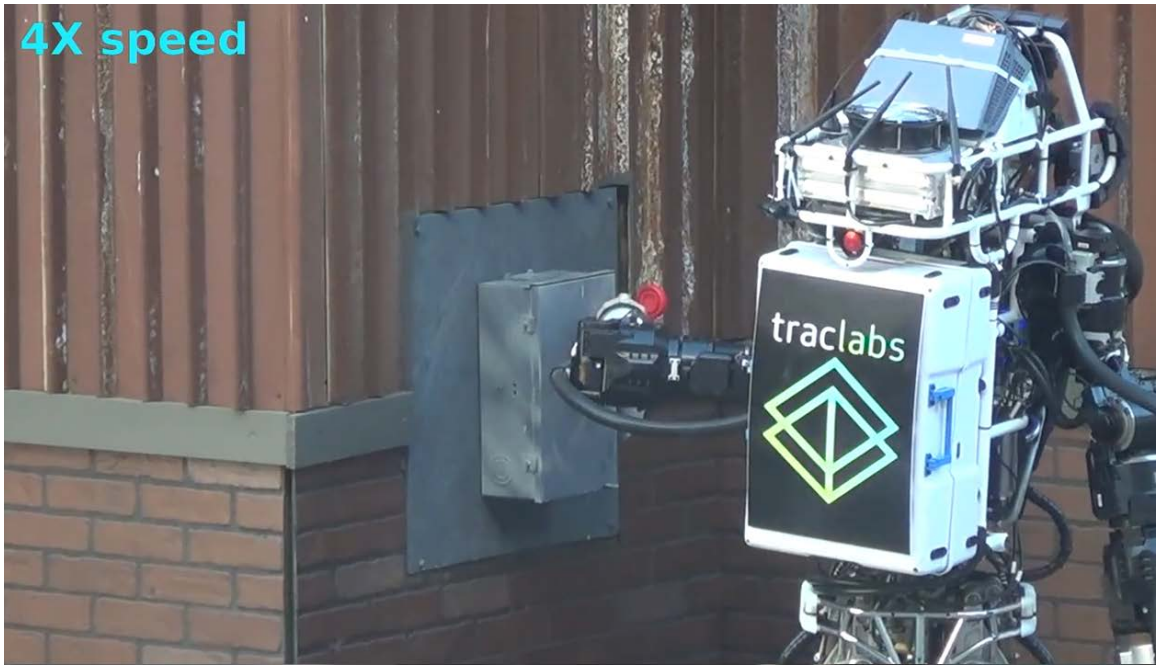

Figure 58: Team TRACLabs accomplishing the mystery task: breaker lever.

4.3.6 Mystery task

We had no trouble with this task during the First Finals Run – giving us our fifth point at about 49 minutes. See Figure 58.

Note: We had a rudimentary script for the "plug" mystery task for the second Finals run. We tested it successfully a few times, but it took a long time and a lot of Operator assistance, and we might have chosen to bypass it in the interest of time, even if we had made it that far.

Figure 59: Team TRACLabs traversing the rough terrain.

4.3.7 Terrain task

For the traverse script, we had tried stepping both feet onto each brick pair, and also alternating feet (more similar to how people walk). Unfortunately, we were using the alternating footstep pattern during the first Finals run (because it covers ground faster), and forgot to set the swing height high enough to accommodate the intervening high bricks. This caused the foot to drag on those high bricks and the robot to lose its balance and fall over to lean on the wall. Using the slightly slower "double-step" traverse pattern, we would not have had any trouble on the terrain. See Figure 59.

4.3.8 Stair task

We were able to accomplish this task very quickly and somewhat reliably in practice. At the Finals, the organizers built all the task apparatuses on the apron of the racetrack, which sloped about 3-5 degrees down to the right. This was an issue, for instance, when the cars needed to drive up over the lip of the apron to get past the finish line! For some reason, however, the staircases were leveled relative to gravity, which meant that the bottom step was roughly an inch and a half higher at the right end than the left on the course we were on. The precise amount varied between the four test tracks. This caused some difficulty for the stepping controller.

4.3.9 Scoring

Our first Finals run was the better run, and we scored five points. We successfully drove and egressed the vehicle, opened the door and walked through it, turned the valve, and flipped the breaker switch. We should have been able to traverse the terrain as well,

but operator error caused the robot to fall without enough time remaining for a reset. Even had we traversed the terrain successfully, it would not have changed our ranking, as we were close to the sixty-minute time.

5 CONCLUDING REMARKS

Analysis of videos of the other teams indicate that we were able to accomplish the indoor tasks at least as quickly as nearly all of the other teams. This shows off the power of the strategy we employed of adjustable autonomy: combining human-in-the-loop with autonomous scripting.

Three teams scored all eight points: KAIST, IHMC, and CMU. KAIST was the clear winner, finishing roughly six minutes faster than IHMC. Looking at the rankings at the Finals when compared to those at the Trials, we see that we were overtaken by KAIST and WPI, but otherwise the relative ranking of the teams was the same.

Of the Atlas teams, we had both the mildest fall, where we gently leaned on the wall at the end of Day One, and the most catastrophic, where we blew hydraulic lines in a manner that guaranteed that the run was terminated in Day Two.

We are satisfied with 9th place at the Finals. We had working strategies for all eight of the tasks, and feel that with a few more test runs on the real course, and another week or so of tuning development, we could be scoring all eight points in less than an hour.

6 REFERENCES

DARPA (2012) DARPA Robotics Challenge. http://www.darpa.mil/Our_Work/TTO/Programs/DARPA_Robotics_Challenge.aspx.

Bruyninckx, H. and Soetens, P. (2002). Open Robot Control Software. http://www.orocos.org/kdl.

Diankov, R. (2008). OpenRave IKFast Module. http://openrave.org/docs/latest_stable/openravepy/ikfast/.

Dutra, M., Salcedo, I., and Prieto Diaz, L. (2008). New technique for inverse kinematics problem using simulated annealing. In International Conference on Engineering Optimization.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

API	Application Program Interface
AT	Affordance Template
BDI	Boston Dynamics, Inc.
CoM	Center of Mass
DARPA	Defense Advanced Research Projects Agency
DOF	Degrees of Freedom
DRC	DARPA Robotics Challenge
FOV	Field of View
GUI	Graphical User Interface
HRI	Human-Robot Interaction
HUD	Heads-Up Display
IK	Inverse Kinematics
IKFast	A particular Third-Party IK Solver
IMU	Inertial Measurement Unit
KDL	Kinematics and Dynamics Library
Kd-tree	K-dimensional tree (a space-partitioning data structure)
LIDAR	Laser rangefinder
LUT	Lookup Table
MATEC	Multi-Appendage Torque and Environment Contact control suite
MoveIt!	Software for Mobile Manipulation
OCS	Operator Control Station
OI	Operator Interface
OSRF	Open-Source Robotics Foundation (simulation software vendor)
PCL	Point Cloud Library
PID	Proportional-Integral-Derivative Control
Qt	A particular cross-platform application framework
R2	Robonaut 2 (A NASA Humanoid Robot)
RNEA	Recursive Newton-Euler Algorithm
ROS	Robot Operating System
RViz	Robot Visualization Tool
SA	Situational Awareness (refers to cameras used for this purpose)
SA-GD	Simulated Annealing-Gradient Descent
SLAM	Simultaneous Localization and Mapping
SRI	SRI International, Inc. (hand vendor)
SUP	Single UDP Pipe
TCP	Transmission Control Protocol (an internet communication protocol)
UDP	User Datagram Protocol (an internet communication protocol)
VRC	Virtual Robotics Challenge